Enabling Efficient and Effective Data Extraction and Data Labeling



Yu Zhang

Jesus College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity Term 2022

Dedicated to my parents.

Abstract

History has left us with many wonderful visualizations. Historical visualizations capture valuable historical statistics. The original datasets that generate historical visualizations are useful to humanities and social science scholars but are often lost. This thesis starts with the problem of extracting such datasets from historical visualizations. It is impractical to expect purely automatic solutions for data extraction, as historical visualizations feature diverse designs, contain image defects, and the data extraction has low error tolerance. Meanwhile, manual extraction is laborious. It calls for efficient and effective solutions.

To this aim, this thesis proposes a machine-aided approach to data extraction from historical visualizations. We convert data extraction to iterative machine learning processes of carrying out classification tasks. Following the conversion, we propose a generic workflow that features active sampling and default labeling to reduce user effort in classification tasks. Based on the approach, we design a data extraction pipeline and develop an interactive system for data extraction from spatial visualizations. To extract data from diverse historical visualizations, a single data extraction pipeline may not suffice. We design a set of reusable data extraction workflows that can be assembled into various data extraction pipelines. We demonstrate the feasibility of the design through prototypes of two pipelines.

Data extraction can be seen as a particular data labeling task. Thus, we investigate the generalization of the machine assistance approach for data extraction into the scenario of data labeling. Through a coding of the literature, we identify a collection of common modules, which extends the set of machine assistance modules we have designed for data extraction. Based on the identified reusable implementations, we have developed an authoring toolkit that builds in common implementations of the modules. Using the toolkit, developers can visually configure data labeling workflows with no code or less code. We demonstrate using the toolkit to develop diverse systems for data labeling and data extraction.

Quality assurance is a laborious process in data extraction and data labeling to validate the correctness of the resulting data. It is thus critical to optimize user productivity in quality assurance interfaces. We propose using model-based evaluation for quality assurance interfaces in machine learning. We model user routines and operation time costs in quality assurance interfaces. The model-based approach

facilitates efficient estimation of task completion time through simulations for quality assurance interfaces. We demonstrate using model-based evaluation in data extraction and aerial image classification scenarios.

This thesis builds up an interactive machine learning approach to data extraction and data labeling. We demonstrate that imperfect machine-learned models can still be used to assist end-users and improve their productivity in these tasks that are hard to automate.

Thesis Outputs

The research conducted in this thesis has led to the following publications.

Yu Zhang, Bob Coecke, and Min Chen. 2019. *On the Cost of Interactions in Interactive Visual Machine Learning.* In Proceedings of the IEEE VIS Workshop on Evaluation of Interactive Visual Machine Learning Systems. Vancouver, BC, Canada. IEEE.

Yu Zhang, Bob Coecke, and Min Chen. 2021. *MI3: Machine-Initiated Intelligent Interaction for Interactive Classification and Data Reconstruction.* ACM Transactions on Interactive Intelligent Systems 11, 3-4, Article 18 (Aug. 2021), 34 pages.

Yu Zhang, Martijn Tennekes, Tim de Jong, Lyana Curier, Bob Coecke, and Min Chen. 2021. *Using Simulation to Aid the Design and Optimization of Intelligent User Interfaces for Quality Assurance Processes in Machine Learning.* arXiv: 2104.01129 [cs.HC].

Yu Zhang, Yun Wang, Haidong Zhang, Bin Zhu, Siming Chen, and Dongmei Zhang. 2022. *OneLabeler: A Flexible System for Building Data Labeling Tools.* In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI '22). New Orleans, LA, USA. ACM.

While I am the principal author of the research conducted in the thesis, much of the work was accomplished in collaboration with my supervisors, Professor Min Chen and Professor Bob Coecke, and colleagues at Microsoft Research and Statistics Netherlands. Throughout the thesis, I use first-person plural pronouns to reflect their contributions.

Acknowledgements

Throughout my DPhil journey, I have received help and support from many people. First and foremost, I want to thank my supervisors, Professor Min Chen and Professor Bob Coecke, for guiding me through the journey.

From my primary supervisor, Professor Min Chen, I have learned a way of thinking and research over the past four years and a half. I am grateful that his office is always open, and he is always welcoming to discussions. Moreover, he gave me the freedom and opportunities to have long visits and internships during my study.

I want to thank my secondary supervisor, Professor Bob Coecke. Although the research I conducted during my DPhil differs from his primary research focus, he still took time for discussions and gave me much helpful advice.

I would also like to thank my internship mentors and colleagues at Microsoft. In particular, I want to thank Yun Wang, Haidong Zhang, Bin Zhu, and Song Ge. The days we fought together for deadlines overnight in the office became delightful memories.

I want to thank my colleagues at Statistics Netherlands for hosting my visit. In particular, I want to thank Tim de Jong, Martijn Tennekes, Lyana Curier, and Han van Leeuwen. I learned much from them about real-world applications of remote sensing and image processing.

I owe a huge debt of gratitude to my family and friends. Thank you to my parents for their love and support. Thanks are also due to my undergraduate supervisor, Professor Xiaoru Yuan, and friends at Peking University, particularly Guozheng Li and Can Liu, for academic discussions and casual chats. I also want to thank Siming Chen, now a professor at Fudan University, for hosting my visit. Being the only student in the research group working on human-computer interaction during my DPhil, the love and encouragement from my family and friends is my infinite source of energy throughout my solo journey.

Contents

1 Introduction **1**

1.1 Motivation 1
1.2 Research Objectives 2
1.3 Thesis Outline 4

2 Background **8**

2.1 A Brief History of Data Visualization 8
2.1.1 Maps 9
2.1.2 Statistical Charts 11
2.1.3 Narrative Charts 15
2.2 Data Extraction from Visualizations 17
2.2.1 Applications 17
2.2.2 Techniques 20
2.3 Interactive Machine Learning 27
2.3.1 Concepts 27
2.3.2 Systems 30
2.3.3 Algorithms 36
2.4 Usability Evaluation 40
2.4.1 Overview 40
2.4.2 Model-Based Evaluation 41

3 Data Extraction from Historical Visualizations with Machine Assistance **44**

3.1 Introduction 44
3.2 Overview of the Machine-Aided Approach 45
3.2.1 Examples of Technical Challenges 46
3.2.2 A Generic Workflow for Interactive Classification 48
3.3 Instantiating Machine-Aided Workflows in Practice 51
3.3.1 Transforming a Detection Problem to a Classification Problem . . 51
3.3.2 Variations of Workflows 52
3.4 Data Extraction from Historical Visualization Images 54
3.4.1 Block Detection Preprocessing 55
3.4.2 Block Grouping Preprocessing 57

3.4.3 Root Detection Preprocessing . . . 58
3.4.4 Interactive Classification and Active Learning . . . 60
3.5 Evaluation with Simulations . . . 64
3.5.1 Defining Ground Truth . . . 65
3.5.2 Dataset . . . 66
3.5.3 Experiment Design . . . 67
3.5.4 Results . . . 69
3.6 Analyzing Reusable Workflows for Data Extraction . . . 72
3.6.1 Reusable Workflows . . . 73
3.6.2 Composing Workflows . . . 75
3.6.3 Prototypes . . . 78
3.7 Domain Expert Interview . . . 81
3.7.1 Interview Design . . . 81
3.7.2 Results . . . 83
3.8 Conclusion . . . 85

4 A Toolkit for Data Extraction and Data Labeling 88
4.1 Introduction . . . 88
4.2 Design Requirements . . . 90
4.2.1 Pilot Study . . . 90
4.2.2 Design Principles and Requirements . . . 91
4.3 Analyzing Reusable Modules for Data Labeling . . . 93
4.3.1 Methodology . . . 93
4.3.2 Process . . . 95
4.3.3 Results . . . 99
4.3.4 Common Modules . . . 100
4.3.5 Module Composition Constraints . . . 102
4.4 Authoring Toolkit . . . 103
4.4.1 System Architecture . . . 104
4.4.2 Visual Programming of Workflows . . . 106
4.4.3 Built-in Modules . . . 107
4.4.4 Customization . . . 110
4.4.5 Workflow Compiler . . . 112
4.5 Usage Scenarios . . . 112
4.5.1 A Gallery of Basic Labeling Systems . . . 113
4.5.2 Customizing a Webpage Data Type . . . 113
4.5.3 Machine-Aided Multi-Task Text Labeling . . . 115
4.5.4 Mixed-Initiative Image Classification . . . 117
4.5.5 Data Extraction from Historical Visualizations . . . 119
4.6 User Study with Developers . . . 121

4.6.1 Experimental Design . . . 121
4.6.2 Results . . . 122
4.7 Discussion . . . 125
4.8 Conclusion . . . 128

5 Model-Based Evaluation of Label Quality Assurance Interfaces 129
5.1 Introduction . . . 129
5.2 Background . . . 131
5.2.1 Data-Centric Applications . . . 131
5.2.2 Grid-Based Interfaces . . . 133
5.3 Overview of the Simulation Approach . . . 135
5.3.1 A Routine Task Workflow . . . 136
5.3.2 A Simulation Model . . . 138
5.3.3 Modeled and Simulated Factors . . . 139
5.3.4 Scope and Assumptions . . . 140
5.4 The Factors of Interface Layout and Application . . . 141
5.4.1 Datasets to be Quality-Assured . . . 141
5.4.2 Conceptual Model for Simulation . . . 142
5.4.3 Estimating Operator Time Costs . . . 143
5.4.4 Simulations and Observations . . . 149
5.5 The Factor of User Label Strategy . . . 150
5.5.1 Conceptual Model for Simulation . . . 150
5.5.2 Estimating Operator Time Costs . . . 153
5.5.3 Simulations and Observations . . . 153
5.6 The Factor of Default Label Accuracy . . . 155
5.6.1 Simulations and Observations . . . 155
5.7 The Factor of Rank Method . . . 156
5.7.1 Conceptual Model for Simulation . . . 157
5.7.2 Simulations and Observations . . . 158
5.8 Summary of the Simulation Model . . . 159
5.9 Discussion . . . 162
5.10 Conclusion . . . 163

6 Conclusion 165
6.1 Contributions . . . 165
6.1.1 Efficiency and Effectiveness . . . 165
6.1.2 Generalization . . . 166
6.1.3 Evaluation and Optimization . . . 167
6.2 Future Work . . . 168

Appendices

A Graph-Theoretic Constraints on Data Labeling Workflows **173**
A.1 Notations . . . 173
A.2 Constraints . . . 174

B User Study Instructions for the Toolkit **176**
B.1 Task 1 (Image Segmentation) . . . 176
B.2 Task 2 (Image Classification) . . . 176
B.3 Task 3 (Machine-Aided Image Classification) . . . 177
B.4 Task 4 - Option 1 (Customize Data Type) . . . 178
B.5 Task 4 - Option 2 (Reproduce Your Tool) . . . 179

C Estimating Operator Time Costs in Quality Assurance Interfaces **180**
C.1 Experiment . . . 180
C.1.1 Apparatus . . . 180
C.1.2 Procedure . . . 181
C.2 Initial Estimations of Operator Time Costs . . . 183
C.3 Modeling and Reestimation of Operator Time Costs . . . 183
C.3.1 Model $\mathbf{t_{new}}$ . . . 183
C.3.2 Reestimate $\mathbf{t_{view}}$ and $\mathbf{t_{single}}$ by $\mathbf{t_{new}}$ Model . . . 187
C.3.3 Model Reestimated $\mathbf{t_{single}}$ . . . 189
C.3.4 Reestimate $\mathbf{t_{view}}$ by $\mathbf{t_{new}}$ and $\mathbf{t_{single}}$ Models . . . 195
C.3.5 Model Reestimated $\mathbf{t_{view}}$ . . . 196
C.4 Final Estimations of Operator Time Costs . . . 199

References **212**

"It gives a simple, accurate, and permanent idea, by giving form and shape to a number of separate ideas, which are otherwise abstract and unconnected." [211]

— William Playfair

1 Introduction

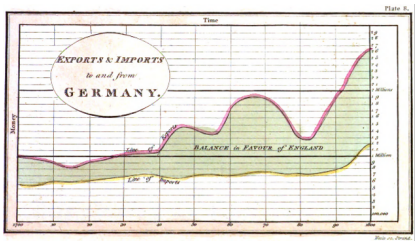
1.1 Motivation

Data visualizations represent data graphically. They have long been a favorite technique for analyzing and communicating data. History has left us with many wonderful *historical visualizations*. For example, the map collection by David Rumsey alone includes thousands of historical statistical charts and thematic maps [231]. Some of the well-known historical visualizations include:

- William Playfair's time series charts of trade balance (1786) [211], bar chart of Scotland's imports/exports (1786) [211], and pie chart of Turkish Empire's land holdings (1801) [213];
- Charles Joseph Minard's flow map of road traffic (1845) [187], and flow map of Napoleon's Russian campaign (1869) [190];
- John Snow's cholera map (1854) [250];
- Florence Nightingale's radial charts (1858) [198, 199].

The original data records that generate historical visualizations are valuable to humanities and social sciences scholars but are often lost over time. Historical visualizations are usually the only information source that captures these historical datasets.

Data extraction is to decode the datasets from the scanned bitmap images of historical visualizations, as shown in Fig. 1.1. Data extraction facilitates downstream applications. Its outcomes can be used to index visualizations by search engines [45, 57] and facilitate data-related query answering [139]. Data extraction is also an essential step to making the visualizations accessible to visually impaired audiences through



Year	Export to Germany	Import from Germany
1700	1 million pounds	0.57 million pounds
...	...	...
1800	2.63 million pounds	1.2 million pounds

Figure 1.1: What is data extraction: Data extraction takes the bitmap image of a visualization as input and outputs the dataset that generates the visualization. For example, given a scanned image of Playfair's line chart on the left [210], the data table on the right shows the outcome of data extraction.

screen readers [63]. In addition, the extracted dataset can be used for data analysis. For example, the economic data encoded in William Playfair's line charts [211] can be useful to cliometricians studying international trade in the 18th century.

Manually extracting datasets from historical visualizations is laborious as it requires tedious manual measurements. This thesis is motivated by the problem of data extraction from historical visualization images and aims to provide a data extraction approach alternative to manual extraction.

1.2 Research Objectives

There are various technical *challenges* in designing computerized approaches to data extraction from historical visualizations:

- **C1** **Diverse designs**: Historical visualizations are diverse in their design because of the early years' lack of standards and conventions. Due to the diversity, handcrafted rules to extract data from one historical visualization may not readily generalize to another.
- **C2** **Image defects**: The scanned images of historical visualizations contain noise from various sources, such as imprecision in printing or hand-drawing and the deterioration of the storage media over time. It is hard to derive automatic algorithms to bypass all the noise.
- **C3** **Low error tolerance**: The extracted data is expected to serve as an accurate benchmark reused in various scenarios.

Machine learning can be used for data extraction, but needs to tackle sparse data, labeling cost, and quality control, which are general challenges in machine learning applications.

- **C4** **Sparse data**: Machine learning techniques usually assume there are numerous similar unlabeled data to be processed by reusing a trained model. However, the

diversity of historical visualizations (**C1**) reduces the opportunities to reuse a trained model. Finding cost-effective ways to train and use disposable models is essential in data-sparse applications.

C5 **Labeling cost**: Supervised machine learning relies on labeled datasets. Meanwhile, crafting labels for data can be labor-intensive and costly. For example, manually crafting a structured dataset for a historical visualization may take hours. It is critical to reduce the labeling cost and improve data annotators' productivity.

C6 **Quality control**: Machine-learned models make erroneous decisions sometimes, especially when trained with sparse, biased, uncertain, conflicting, or dated data. Such errors can be ignored by the end-users with little cost in some applications, such as product recommendations. In other applications, such as when generating benchmark datasets, quality control of model predictions is necessary. Data extraction from historical visualizations is one of these applications (**C3**).

Introducing user interventions for quality control may result in accurate data extraction outcome. Nevertheless, the quality control interface is labor-intensive and needs to optimize user productivity. For the interactive approach to be meaningful, the amount of user effort needs to be lower than with manual extraction.

C7 **Evaluate design variations**: Carrying out quality control tasks requires users to repeatedly verify model predictions, which is laborious and time-consuming. Thus, it is critical to evaluate design variations to optimize the user interface to minimize user effort in these routine tasks.

To address the challenges in data extraction from historical visualizations, we focus on the following *objectives*:

O1 **To extract data efficiently and effectively**: Data extraction from historical visualization is hard to automate fully because of the diverse designs (**C1**), the low image quality (**C2**), and the low error tolerance in extracted data (**C3**). Human intervention is necessary, requiring an interactive system. Meanwhile, it is beneficial to improve the efficiency of the interactive data extraction process through machine assistance. Specifically, an appropriate approach may be to train mediocre and disposable models incrementally, not for reuse, due to the sparse data (**C4**), but to assist the user in data extraction.

O2 **To generalize to other data-sparse scenarios**: While we start with the scenario of data extraction from historical visualizations, we aim to derive transferrable knowledge to other scenarios. Thus, it is necessary to abstract the data extraction problem to relate it to other scenarios. We specifically consider the general scenario of data labeling, which also faces the challenge of labeling cost (**C5**).

O3 To evaluate and optimize with simulations: As we focus on reducing user effort in data extraction, it is necessary to evaluate and optimize the amount of required user effort. We primarily focus on evaluating interfaces where the user conducts quality control of model predictions (**C6**). We utilize simulations of user operations to explore and optimize design variations (**C7**).

It is hard to automate data extraction fully. Human needs to be involved in solving the problem. Meanwhile, the machine may assist the user in data extraction. It is thus helpful to develop interfaces to join human and machine intelligence in solving data extraction problems. The machine intelligence in assisting is always imperfect, as otherwise, the concerned problem would have been solved automatically. Meanwhile, machine assistance may improve the data extraction process for the user in various aspects, such as improving user productivity.

With the prosperity of machine learning (ML) techniques, recent years have witnessed an increasing trend in human-computer interaction (HCI) and visualization (VIS) research communities to build interactive systems that involve iterative training of ML models to assist the user in tasks. This body of work is referred to as interactive machine learning (IML). To address the research objectives, this thesis proposes an interactive machine learning approach to data extraction from historical visualization images and its generalization and evaluation.

1.3 Thesis Outline

This thesis researches interactive systems and techniques to address the research objectives. The following outlines the thesis content and contributions.

Chapter 2 introduces historical visualizations and reviews the literature on data extraction, interactive machine learning, and usability evaluation. We start with a chronology of historical visualization inventions and introduce applications and techniques for data extraction. This thesis proposes a machine-aided approach for data extraction from historical visualizations in Chapter 3 and its generalization to data labeling scenarios in Chapter 4. Our approach falls within the scope of interactive machine learning. Thus, we review techniques in interactive machine learning and related concepts to contextualize our contribution. We also review usability evaluation methods and focus on model-based evaluation methods to contextualize our simulation-based evaluation approach in Chapter 5.

Chapter 3 proposes a generic machine-aided approach to data extraction from historical visualization images. To support **O1 (efficiency and effectiveness)** in data extraction, we convert data extraction into an iterative machine learning process

of gathering classification labels. For effectiveness, we need human-in-the-loop to ensure data extraction accuracy. Within the interactive machine learning process, the machine raises questions to the user to gather information needed for data extraction. For efficiency, we want to reduce the amount of user effort in the data extraction process. Our approach exploits active learning to focus user effort on challenging cases and default labeling to provide suggestions. The model used for active learning and default labeling iteratively learns from the user to improve its performance.

In our approach, the iterative process where the machine raises questions and the user provides answers forms data labeling sessions in active learning. The user interaction plays a role in labeling unlabeled objects, correcting errors made by the current model being actively trained, and monitoring the training process to see if the model predictions have reached the quality level required.

We design a pipeline that sequentially assembles data extraction steps for visual object detection, visual object grouping, and visual attribute measurement. Each step is designed following our interactive machine learning approach. We develop an interactive intelligent system that implements the pipeline to support data extraction from spatial data visualizations. To extract data from diverse historical visualizations, a single data extraction pipeline may not suffice. We design a set of reusable building blocks that can be assembled into data extraction pipelines and demonstrate the feasibility of the design by implementing two pipelines as prototype systems.

We compare variations of data extraction approaches through a simulation-based evaluation that compares the number of required user operations in data extraction. Through interviews with domain experts, we identify usage scenarios of data extraction from historical visualizations. The scenarios include studying the spatial correlation of landmarks, distortions in historical visualizations, and dating the creation time.

The research in this chapter is published in ACM Transactions on Interactive Intelligent Systems [297] and was presented at ACM Conference on Intelligent User Interfaces. The contributions of this chapter include:

- a **generic approach** to data extraction from historical visualizations that converts it to classification tasks and provides machine assistance,
- a demonstration of actualizing the approach through the design of reusable data extraction **workflows** and the implementation of data extraction **systems**, and
- a quantitative **simulation-based evaluation** of the approach in three workflows and a **domain expert interview** on the usage scenarios of data extraction.

Collecting labeled datasets is a critical step in supervised machine learning applications. Different applications require various data labeling systems, which are costly to develop. Chapter 3 focuses on the scenario of data extraction from historical

visualizations. It can be seen as an instance of data labeling tasks that requires collecting structured labels. We investigate the generalization of the machine assistance approach in the general scenario of data labeling.

Chapter 4 introduces an authoring toolkit to develop interactive machine learning systems for data labeling. The toolkit builds on a conceptual framework of reusable modules in data labeling systems and module composition constraints. Using the toolkit, a developer can develop diverse interactive machine learning systems for data labeling, including data extraction systems, by visually configuring workflows with no code or less code.

We identify building blocks, referred to as modules, of interactive machine learning systems for data labeling by coding the workflows in the literature. Each module is a machine or a human computation step, such as default labeling or user sampling. Through the coding process, we identify eight types of modules: interactive labeling, data object selection, model training, feature extraction, default labeling, quality assurance, stoppage analysis, and label ideation. The modules can be implemented as algorithm components and interface widgets, which involve machine computation and human computation. The identified reusable building blocks inform the development of an authoring toolkit for building data labeling systems.

We demonstrate usage scenarios of the toolkit where we build diverse data labeling systems with different visually configured workflows. Data extraction tasks can be reduced to classification tasks, which are instances of data labeling tasks. Using the toolkit, we demonstrate reproducing Chapter 3's data extraction system. The toolkit addresses **O2 (generalization)** and demonstrates the feasibility of generalizing the machine assistance modules, such as the active sampling and default labeling, introduced in Chapter 3 into general data labeling scenarios.

The research in this chapter is published in Proceedings of the ACM CHI Conference on Human Factors in Computing Systems [299] and is open-sourced at `https://github.com/microsoft/OneLabeler`. The contributions of this chapter include:

- a conceptual framework that illuminates **reusable modules** and composition constraints in data labeling systems,
- an **authoring toolkit** based on the framework to support the easy building of diverse data labeling systems, and
- extensive **usage scenarios** to demonstrate the capability of the toolkit in creating diverse systems, including a data extraction system, as well as a **user study** to validate its usability for developers.

Previous chapters have investigated interactive intelligent systems for data extraction and data labeling where user intervention is necessary for accurate data extraction

and data labeling results. When designing data extraction systems and data labeling systems, one needs to make various design choices. It is necessary to evaluate and optimize the design choices to reduce the cost of user interventions. Among the various forms of user interventions, the quality assurance of machine predictions is probably the most tedious and time-consuming for the user, as it needs to be repeated numerous times.

Chapter 5 proposes using model-based evaluation to estimate task completion time in data extraction and data labeling systems through simulations. While Chapter 4 focuses on spanning the design space of data extraction and data labeling systems, this chapter focuses on efficiently evaluating and optimizing systems within the space to address **O3 (evaluation and optimization)**.

We analyze commonalities in these systems and summarize a workflow of user and machine operations for accomplishing data extraction and data labeling tasks. Based on the workflow, we design a simulation model that incorporates various factors as simulation parameters, such as user operation time costs and the accuracy of machine-predicted default labels. We demonstrate a procedure to estimate and model the user operation time costs.

The simulation approach supports the efficient evaluation of design variations of quality assurance interfaces. While the task completion time estimated by simulation is not as conclusive as user study results, it is a more efficient alternative suitable for early-stage design iterations.

The research in this chapter is available on ArXiv [298] and is partially published in Proceedings of the IEEE VIS Workshop on Evaluation of Interactive Visual Machine Learning Systems [296]. The contributions of this chapter include:

- a simulation **model** of users' routine operation sequence and operation time costs in quality assurance interfaces, and
- an **application** of the model to evaluate and optimize quality assurance interfaces for data extraction and aerial image classification.

Chapter 6 concludes the thesis and discusses future work motivated by the thesis.

Summary: Imperfect machine-learned models can still be used to assist end-users and improve their productivity in tasks that are hard to automate. Such tasks include data labeling and its instance: data extraction from historical visualizations.

"The dominant principle that characterizes my graphic tables and my figurative maps is to immediately appreciate by the eye, as much as possible, the proportions of the numerical results." [188] (translated from French)

— Charles Joseph Minard

2 Background

This thesis starts with the problem of data extraction from historical visualizations. Section 2.1 introduces a history of visualizations and gives examples of historical visualizations. Section 2.2.1 introduces applications supported by data extraction. The two sections motivate the thesis.

The technical approach of the thesis (see Chapters 3 and 4) specifically falls in the category of intelligent user interfaces and interactive machine learning, as we exploit assistance from machine-learned models for data extraction and data labeling. Section 2.2.2 reviews techniques for data extraction from visualizations. Section 2.3 reviews the literature on intelligent user interfaces and specifically focuses on interactive machine learning for data labeling.

Chapter 5 proposes a simulation approach based on GOMS to evaluate data extraction and data labeling systems. We review evaluation techniques and focus on model-based evaluation methods in Section 2.4.

2.1 A Brief History of Data Visualization

While contemporary visualizations are mainly created with computerized toolkits, visualization has a long history that significantly predates the computer age. This section chronologically introduces the origins of classical visualization chart types with examples of well-known historical visualizations as listed in Table 2.1. We group the chart types into three categories: maps, statistical charts, and narrative charts. We also reflect on the historical background that leads to the development of visualization techniques.

Table 2.1: The historical visualizations introduced in this section. The italicized name refers to an excerpt of the original title given by the visualization's author.

Time	Name	Chart Type	Author
c. 11000 BCE	Navarra stone map [266]	geographical map	unknown
c. 1600 BCE	Senenmut star map [251]	star map	unknown
c. 150 BCE	Fangmatan paper map [125]	geographical map	unknown
c. 91 BCE	*Yearly Chronicle of the Six States* [249]	timeline	Qian Sima
c. 50 BCE	Dendera zodiac [228]	star map	unknown
c. 950	planetary movement through zodiac [265]	line chart	unknown
c. 1350	functional relations [202]	bar chart	Nicole Oresme
1686	*An Historical Account of the Trade Winds* [111]	thematic map	Edmond Halley
1765	*A Chart of Biography* [217]	timeline	Joseph Priestley
1769	*A New Chart of History* [218]	timeline	Joseph Priestley
1786	*Exports & Imports to and from all North America* [211]	line chart	William Playfair
1786	*Exports & Imports of Scotland* [211]	bar chart	William Playfair
1801	*The extent, the population and revenues* [212]	pie chart	William Playfair
1826	*Instruction populaire de la France* [79]	choropleth map	Charles Dupin
1833	scatterplot for curve fitting [119]	scatterplot	John Herschel
1833	*Influence de l'Age* [107]	parallel coordinates	André-Michel Guerry
1852	*Temperature and Mortality of London* [86]	radial line chart	William Farr
1855	cholera map [250]	thematic map	John Snow
1858	*Diagrams Representing the Relative Mortality* [198]	radar chart	Florence Nightingale
1869	map of Napoleon's Russian campaign [190]	thematic map	Charles Joseph Minard
1877	*Linien-Diagramme im Kreise* [184]	radar chart	Georg von Mayr
1914	*Number of Passengers Carried on the Railroads* [39]	ISOTYPE	Willard C. Brinton
1930	*Entwicklung der Kartofffelproduktion seit 1860* [193]	ISOTYPE	Otto Neurath

2.1.1 Maps

The oldest visualizations are maps that encode spatial information. They are indispensable for almost every application that concerns transportation or spatial analysis. Due to their long history and wide use, enormous historical maps are available nowadays. For example, Old Maps Online [151] has indexed over 400000 maps.

Geographical Maps

Geographical maps show location information of administrative regions and visible geographical features, such as the distribution of water bodies. They are the earliest known form of visualization. While it is hard to ascertain the inventor of maps, archaeological evidence suggests the history of geographic maps dates back to the engraved sandstone at least the paleolithic age more than 13000 years ago [266]. The early geographical maps are illustrative and thus may not be considered visualizations. It is hard to infer distances between locations from these early maps reliably.

Like the paleolithic age map, early maps are usually engraved on stone blocks or clay tablets. Portable maps made of paper appeared much later. The earliest known paper-based map was the Fangmatan map created in the second century BCE [125]. This map is also the earliest extant paper fragment.

The invention of the geographic coordinate system brings rigor to mapping. This invention is usually credited to Eratosthenes of Cyrene, who lived around the third century BCE for his work *"Geographika"*. Later, around 150 CE, Ptolemy wrote *"Ge-*

ographike Hyphegesis" [219], which integrates the coordinate systems. Ptolemy's book provides a guide on map drawing for the part of the world known to the Greco-Roman societies and the latitude and longitude coordinates for thousands of places.

Star Maps

In addition to geographic maps capturing terrestrial features, star maps capturing celestial features also has a long history. The earliest star chart dates to at least the 16th century BCE [251].

The Dendera zodiac is a famous star map sculptured on the ceiling of a chapel in the Dendera Temple complex. It dates to around 50 BCE and is one of the earliest complete star maps [228]. The star map shows zodiac signs with constellations represented in a mixture of Egyptian and Greco-Roman iconographic forms.

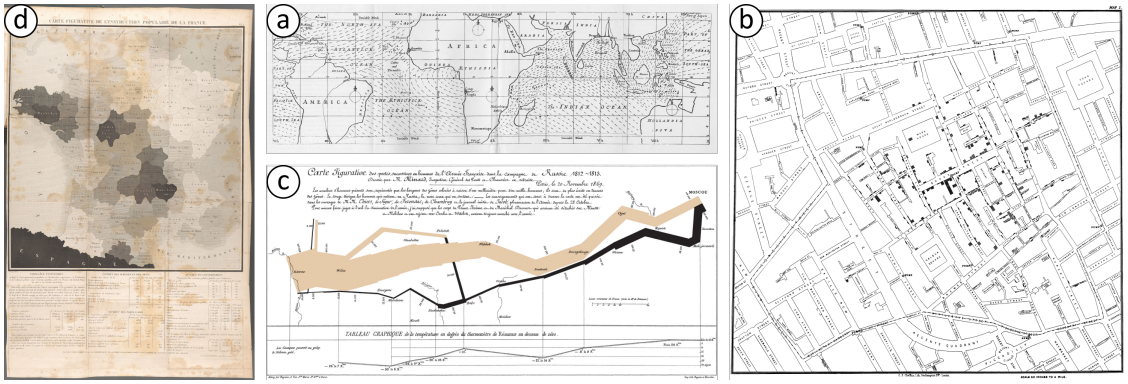


Figure 2.1: Historical thematic maps: (a) Edmond Halley's 1686 map showing directions of trade winds and monsoons [111]. (b) John Snow's 1855 map showing the distribution of cholera fatalities published in the book *"On the Mode of Communication of Cholera"* [250]. (c) Charles Joseph Minard's 1869 map showing successive losses of soldiers in Napoleon's Russian campaign [190]. (d) Charles Dupin's 1826 map showing the rate of male children in primary schools in each region [79].

Thematic Maps

Compared with geographical maps, thematic maps depict additional properties that are not naturally visible, such as wind direction or demographic data. Edmond Halley's 1686 map (Fig. 2.1(a)) shows the direction of trade winds and monsoons with dashed lines [111]. It is the first known weather map and possibly the first thematic map.

John Snow's 1855 map (Fig. 2.1(b)) is a famous historical thematic map [250]. It shows the spatial distribution of fatalities in the 1854 London Cholera Outbreak in relation to a tainted pump. Snow discovered that the spread of cholera was due to a tainted pump on Broad Street.

Charles Joseph Minard's 1869 map (Fig. 2.1(c)) was praised as "the best statistical graphic ever produced" by Tufte [261]. It shows successive losses of soldiers

in Napoleon's Russian campaign between 1812 and 1813 [190]. The map embeds a Sankey diagram that visualizes army size and a line chart showing the temperature. Textual labels along the line present temporal information. Note that the term "Sankey diagram" is named after Matthew Henry Phineas Riall Sankey for his chart of energy flows in steam engines [144], which was created later than Minard's map.

Choropleth maps are a common form of thematic maps. Charles Dupin published the first known choropleth map in 1826 (Fig. 2.1(d)). The map uses color to encode the rate of male children in primary schools in each administrative department in France [79].

Choropleth maps were then quickly adopted by scholars studying moral statistics, including André-Michel Guerry [107] and Adolphe d'Angeville [69]. Moral statistics concern data related to social pathologies, such as data on crimes, illiteracy, and divorce. The study of moral statistics led to the development of quantitative methods in criminology and sociology.

Thematic maps are diverse in that they can embed various visualization techniques, such as the statistical and narrative charts introduced below. One can easily find thousands of thematic maps in online collections, such as David Rumsey Map Collection [231]. Due to the prevalence of historical thematic maps, Chapter 3 uses historical thematic maps as examples for data extraction. Specifically, we have used John Snow's cholera map (Fig. 2.1(b)) and Charles Joseph Minard's map of Napoleon's march (Fig. 2.1(c)) as examples.

2.1.2 Statistical Charts

It was not until recent centuries that visualizations started to represent data outside the scope of geography and astronomy. Descartes' invention of the rectangular coordinates [73] in 1637 is a fundamental cause of this change. The rectangular coordinates can be seen as an abstraction of geographic and celestial coordinates. It leads to the idea of presenting data capturing invisible properties (e.g., economic growth) in the coordinates, just like presenting visible geographic/celestial features in geographical/celestial coordinates.

Michael van Langren's 1-dimensional plot published in 1644 uses the Cartesian coordinate to show the previous estimations of longitude between Toledo and Rome [160]. According to Friendly et al.'s study, it may be the first visualization of statistical data outside the scope of geographical maps and star maps [95]. Later on, the golden age of visualization started in the late 18th century [93]. Many commonly used statistical charts were invented between the late 18th and early 19th centuries, including bar charts, line charts, pie charts, parallel coordinates, scatterplots, and radar charts. Explicit use of coordinate axes is common in these charts.

Despite the long history of visualization, most of the commonly used visualiza-

tion techniques came into existence after the 18th century. Friendly and Wainer insightfully comment that the development of visualization techniques after the 18th century relates to the rise of "empirical thinking" in the sixteen century [96]. By "empirical thinking", they refer to "the view that many scientific questions could better be addressed by gathering relevant data than by applying even the best abstract or theoretical thinking." [96]. An example of empirical thinking was Newton using Galileo's experimental data to derive the law of motion.

Empirical thinking relies on data. Thus, it necessitates approaches for understanding the data, and visualization comes to help. The well-known historical visualizations covered in the chronology span various domains: geography, astronomy, history, economy, epidemiology, public health, criminology, and sociology. It implies the broad adoption of empirical thinking and data-driven approaches in sciences.

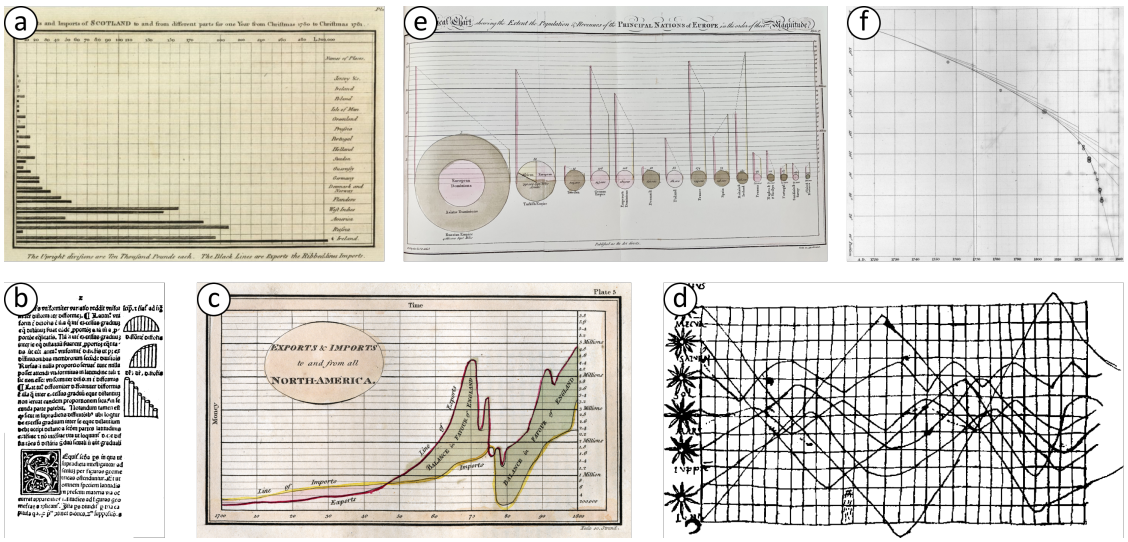


Figure 2.2: Forerunners of common statistical charts: (a) William Playfair's 1786 bar chart showing the distribution of exports and imports of Scotland [211]. (b) Nicole Oresme's fourteen-century illustration of functional relations resembling bar chart [202]. (c) William Playfair's 1786 line chart showing Britain's exports and imports to and from north America [211]. (d) A 10th-century plot showing planetary movement through zodiac [265]. (e) William Playfair's 1801 chart showing the population and revenues of several European countries [213]. (f) John Herschel's 1833 scatterplot for curve fitting [119].

Bar Charts

William Playfair is famous for his invention of various visualization techniques. The invention of the bar chart is widely attributed to him for his bar chart (Fig. 2.2(a)) published in the book *"The Commercial and Political Atlas"* in 1786 [211]. This horizontal grouped bar chart shows the exports and imports of Scotland to and from various places.

Before Playfair, Nicole Oresme, who lived in the fourteen century, also contributed to the invention of the bar chart. In the book *"Tractatus de latitudinibus formarum"* [202] (translates to *"Treatise on latitude of forms"*), his figures (Fig. 2.2(b)) resembling bar charts show the functional relationship between time and velocity. These figures are usually seen as illustrative figures not based on real datasets and thus not categorized as visualizations [95].

Line Charts

In *"The Commercial and Political Atlas"* [211], Playfair extensively used line charts to present and analyze economic statistics, such as national revenues. Playfair's line charts published in 1786 are usually regarded as the earliest line charts [96]. Fig. 2.2(c) shows an example of these line charts. This line chart is somewhat between the line charts and area charts nowadays, with the area between two lines colored. Before Playfair, visual representations similar to line charts have been used on some other occasions, such as the 10th-century plot of planetary movement [97] (Fig. 2.2(d)) and Francis Hauksbee's hyperbolic figure [115].

The 10th-century plot of planetary movement (Fig. 2.2(d)) uses lines to represent the movement of planets through the zodiac [265]. The x-axis corresponds to time or, equivalently, the zodiac, and the y-axis corresponds to the position of planets relative to the zodiac. This historical line chart shows the visible trace of planets, while modern line charts usually show the abstract trace of data values.

Pie Charts

Playfair also published the first known pie chart (Fig. 2.2(e)) in *"The Statistical Breviary"* [213] in 1801. In this chart, each circle's area represents a country's area. The bar on the left of each circle shows the population size. The bar on the right of each circle shows the revenue. For the circle corresponding to the Turkish Empire, Playfair used a pie chart to show the proportion of area in Africa and Europe.

Scatterplot

In 2018, Battle et al. found that bar charts, line charts, scatterplots, and geographic maps are the four most frequently used types of visualizations on the web [21]. Scatterplot is the latest invented type of chart among the four.

John Herschel published the first known scatterplot (Fig. 2.2(f)) in the paper titled *"On the Investigation of the Orbits of Revolving Double Stars"* [119] in 1833. Herschel used a scatterplot to fit curves on the relation between the position angles of double stars (the vertical axis) and time (the horizontal axis). The double circles in the scatterplot represent the data points considered more reliable. The tangent lines show the angular velocities.

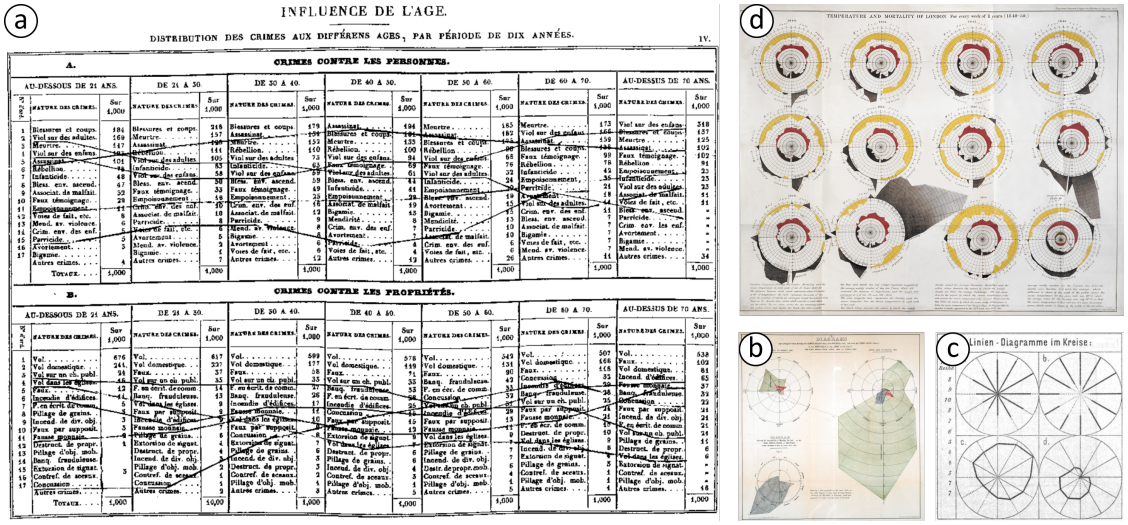


Figure 2.3: Beyond the rectangular coordinate system: (a) André-Michel Guerry's 1833 parallel coordinates showing crime frequencies in different age groups [107]. (b) Florence Nightingale's 1858 radar chart of causes of mortalities [198]. (c) Georg von Mayr's 1877 radar chart titled *"Linien-Diagramme im Kreise"* (translates to *"Line Charts in Circles"*) [184]. (d) William Farr's 1852 radial line chart showing mortalities in relation to temperature [86].

Parallel Coordinates

The common statistical charts in Fig. 2.2 are mostly based on the rectangular coordinate system (except the pie chart in Fig. 2.2(e)). In addition to the rectangular coordinate system, parallel coordinate and polar coordinate systems are also frequently used in visualizations.

André-Michel Guerry published the first known parallel coordinates (Fig. 2.3(a)) in the book *"Essai sur la statistique morale de la France"* in 1833 [107]. This chart shows the crime frequencies of each age group as a ranked list. To highlight the trend, Guerry used lines to connect the same crime across age groups. Compared with this chart, contemporary parallel coordinates replace the ranked lists with axes. Alfred Inselberg developed and popularized the visual design of contemporary parallel coordinates [130].

Radar Chart

Radar charts use the distance from the polar to the points on axes to encode data and connect the points with line segments. They are essentially line charts or area charts in the polar coordinate system.

Florence Nightingale published the first known radar chart (Fig. 2.3(b)) in a report to Royal Commission [198] in 1858. This radar chart titled *"Diagrams Representing the Relative Mortality from Zymotic Diseases, from Wounds, and from All Other Causes"* shows the distribution of mortality causes in each month. The axes of this radar

chart represent the mortalities in each month. The color of the regions notes the causes of mortalities.

Georg von Mayr's 1877 chart (Fig. 2.3(c)) uses a toy dataset to illustrate two design dimensions of radar charts, leading to four design variants [184]. Firstly, the chart may use the distance from the outer circle or the polar to encode data. Secondly, the line segments of the radar chart can be either explicitly plotted or not. The title of von Mayr's chart, *"Line Charts in Circles"*, suggests that radar charts are designed consciously as line charts in the polar coordinate system.

Aside from radar charts, some other historical visualizations use the polar coordinate system. For example, Nightingale's 1858 chart titled *"Diagram of the Causes of Mortality in the Army in the East"* [199] is a radial bar chart. William Farr's 1852 chart (Fig. 2.3(d)) titled *"Temperature and Mortality of London, for every week of 11 years"* [86] is a small multiple of radial line charts.

2.1.3 Narrative Charts

In addition to visualizations created for analyzing data, many narrative visualization techniques have been developed for communicating data. The following introduces the earliest timelines and ISOTYPE, typically categorized as narrative visualization techniques.

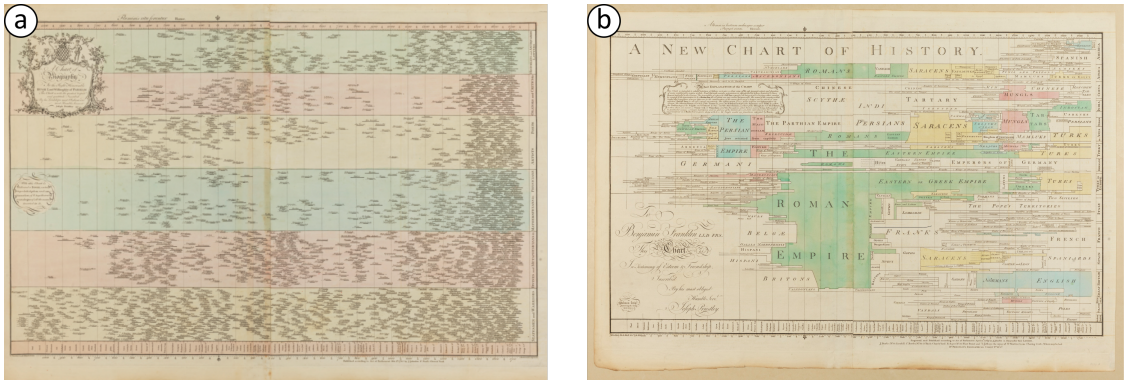


Figure 2.4: Joseph Priestley's timelines: (a) Timeline titled *"A Chart of Biography"* published in 1765 showing lifespans of historical figures [217]. (b) Timeline titled *"A New Chart of History"* published in 1769 showing lifespans and territories of empires [218].

Timeline

Timelines present the temporal order of events, providing an overview of the narrative [239]. Qian Sima published the first known timelines in the book *"Shiji"* [248] (translates to *"Records of the Grand Historian"*) around 91 BCE and established a convention of using timelines in official history. Fig. 3.17 shows a timeline in *"Shiji"* titled *"Yearly Chronicle of the Six States"*.

Joseph Priestley published a well-known timeline (Fig. 2.4(a)) titled *"A Chart of Biography"* in 1765 [217]. It shows historical figures' lifespans between 1200 BCE and 1800 CE. Later, Priestley published another timeline (Fig. 2.4(b)) titled *"A New Chart of History"* in 1769 [218]. It shows the durations, locations, and passing of power of empires. Priestley's timelines were created for the educational purpose of allowing students to trace the dependence of historical events in world history.

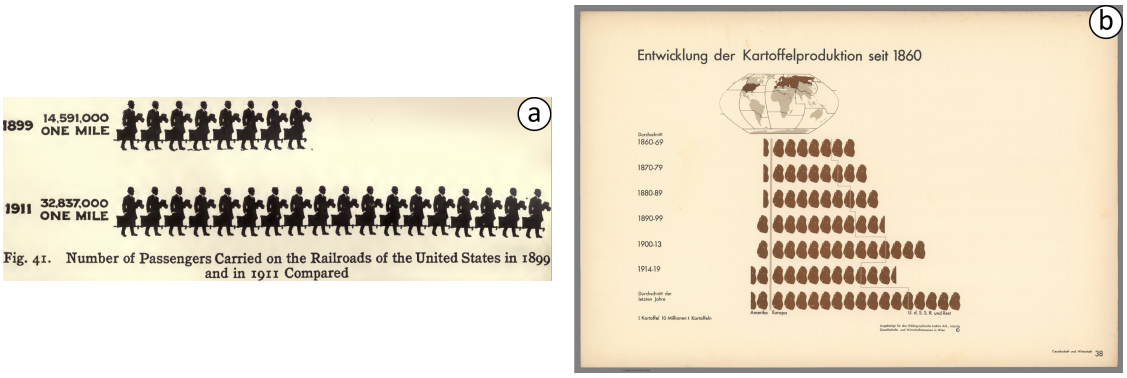


Figure 2.5: Historical ISOTYPE: (a) Willard C. Brinton's 1914 ISOTYPE showing the railroad passenger growth [39]. (b) Otto Neurath's 1930 ISOTYPE showing potato production growth [193].

ISOTYPE

The ISOTYPE method uses repeated pictorial symbols to encode values. It is referred to as an instance of the "concretization" narrative pattern that uses concrete objects to illustrate abstract concepts [239]. Willard C. Brinton published the first known ISOTYPE (Fig. 2.5(a)) in 1914 [39]. The chart uses repeated pictographs to show the approximate proportion of railroad passengers in 1899 and 1911.

The name ISOTYPE (International System Of TYpographic Picture Education) was given by Otto Neurath, who further developed and popularized ISOTYPE. In 1930, Neurath published a collection of a hundred charts, most of which used ISOTYPE, in the book *"Gesellschaft und Wirtschaft"* [193]. The chart in Fig. 2.5(b) is one of the one hundred charts. It shows the growth of potato production since 1860. A potato symbol represents the production of 10 million tons of potatoes.

As the founding director of the Social and Economic Museum of Vienna, Neurath wanted the museum to serve a teaching purpose. He postulated that "to remember simplified pictures is better than to forget accurate figures" [194]. ISOTYPE was the principal approach Neurath utilized to help the audience "remember simplified pictures". Later experiments provide evidence for Neurath's intuition. Haroz et al.'s study suggests that ISOTYPE helps the user remember information during demanding tasks and motivates the user to inspect the visualization more closely [113].

2.2 Data Extraction from Visualizations

Definition of Data Extraction: Denote a dataset as D. Data visualization uses a visual mapping F to map the dataset into a chart $F(D)$ in the image space. Data extraction is an inverse process of data visualization that extracts D given $F(D)$.

What counts as data in historical visualizations and the storage format of the extracted data is up to the user's application requirement. Throughout the thesis, we do not count conventional map features, such as buildings and road names, as part of the data. The extraction of conventional map features is extensively studied in map digitization [62], and these map features are usually not the central information that historical visualizations aim to convey.

This thesis concerns the problem of unlocking the dataset from historical visualizations through data extraction. The following reviews data extraction's downstream applications and techniques.

2.2.1 Applications

Through data extraction, various downstream applications are facilitated. For example, historians may be interested in performing further quantitative analysis on the historical datasets. Kerle and de Leeuw demonstrate such an application where they analyze the population data extracted from a population map [145]. Below, we introduce such application scenarios of data extraction, including information retrieval, chart redesign, and visualization usage pattern analysis.

Some of the work introduced below takes visualizations stored in structured images (e.g., SVG and XML) as input to the data extraction process and is not necessarily focused on historical visualizations. Meanwhile, the applications they demonstrate apply to the data extraction scenarios where bitmap images of historical visualizations are the input.

Information Retrieval

Data extraction facilitates indexing and query answering by information retrieval systems such as digital libraries and search engines.

- **Indexing**: Data extraction provides rich information for indexing. Chen et al. propose DiagramFlyer [57], a search engine for statistical charts in PDFs, that supports query expansion and sorting by TF-IDF scores. They extract metadata from visualizations for indexing, such as labels, scales, titles, legends, captions, and chart types. Ray Choudhury et al. demonstrate a similar application of indexing line chart images in documents [224]. Poco and Heer proposed a machine learning workflow to extract a specification in JSON format from visualizations and suggest using the specification for indexing by search engines [215].

- **Query answering**: Extracted data make possible advanced search functions, such as query answering based on the visualizations' content. Carberry et al. propose using Bayesian Network to infer patterns in bar charts stored as XML [45]. It supports query answering on predefined patterns, such as maximum, increasing trend, and rank. For line chart images, FigureSeer [247] extracts data with a CNN and text with OCR engines. It supports answering queries that match a query template.

Chart Redesign

Chart redesign is beneficial when an existing visualization is ineffective for the concerned usage scenario. It enables the audiences to view the dataset from another perspective. For historical visualizations, it may be interesting to redesign them with contemporary visualization techniques. The redesign may assign a new visual encoding to the underlying dataset (*visual encoding redesign*), or attach new interaction functions to the visualization (*interaction redesign*). Data extraction is a preprocessing to facilitate chart redesign.

Visual Encoding Redesign With the extracted dataset, it is possible to redesign a presentation of the dataset. The redesign may serve the purpose of improving the accessibility of the visualization or the effectiveness of the visual encoding following guidelines. The redesign may also be carried out by end-users creatively in authoring toolkits.

- **Accessibility**: Visual encoding redesign is essential to promote the accessibility of the visualization. For audiences with color vision deficiencies, redesigning the color mapping may highlight initially unnoticeable details. Kuhn et al. propose a recoloring technique to make colors in visualization images differentiable to audiences with color vision deficiencies [157]. While highlighting the initially unnoticeable details with recoloring, their approach aims to preserve the image's original color as much as possible to preserve its naturalness. For visually impaired audiences, data extraction is necessary for redesigning an audio or haptic representation of the dataset. Choi et al. use a collection of models for chart type classification, text extraction, and data extraction based on chart type [63]. They build a browser plugin that converts the extracted data into HTML tables so the audience can access the data with screen readers.

- **Visual encoding improvement**: Visual encoding redesign may also improve the effectiveness of the visualization for audiences with no vision disabilities. Graphical perception research establishes rankings on the effectiveness of visual channels [66]. For example, crowdsourced experiments show that using position

channels to encode numeric values leads to lower errors on proportion judgment than using angle or area channels [116]. Meanwhile, visualizations created by practitioners may not necessarily follow the guidelines implied by the graphical perception experiment results. Following these guidelines, some data extraction systems automatically generate redesigns with improved visual encoding. ReVision extracts data from bitmap images of pie charts and bar charts [236]. It generates a collection of alternative designs ranked following guidelines on the effectiveness of visual channels summarized by Mackinlay [181]. Poco et al. propose a technique to extract data encoded in the color channel of visualizations by parsing color legends [216]. After data extraction, the user can change the color mapping to improve the visualization's readability, such as replacing the rainbow color scheme with the red-white-blue color scheme. Fan et al. propose a system to semi-automatically annotate and revise deceptive line chart images [85]. They extract data and specifications from line chart images and check them against design rules, such as requiring that y-axes should not be truncated or inverted.

- **Creative redesign**: Some visualization and infographics authoring toolkits integrate data extraction functionalities for the users to creatively redesign the visualizations. Kong and Agrawala use data extracted from visualization images to enable users to create various types of information overlays [153]. The overlays show reference structure (e.g., grid lines), highlight, redundant encoding (e.g., overlay a line chart on a bar chart), summary statistics, and annotations. Transmogrification adapts skeleton-based image deformation algorithms in computer graphics to enable users to carry out deformation-based visualization redesign, such as coordinate transformation. [41]. Harper and Agrawala propose a reverse engineering tool to extract data, visual objects, and their mappings from D3.js generated visualizations [114]. It enables the users to restyle the visualizations by removing, changing, or adding mappings from data to visual attributes of visual objects. iVoLVER is a visual programming toolkit for visualization authoring [185]. It enables users to use data encoded in position, length, or area channels of existing visualizations as data sources to create new visualizations. To unlock data from existing visualizations, the users need to manually configure how to parse the data from visual attribute values.

For historical visualizations, using contemporary visualization techniques to visualize the original historical datasets may provide novel insights. Kerle and de Leeuw demonstrate such an application where they extract data from a population map created in 1962 and redesign a 3D population map with the dataset [145].

Interaction Redesign Aside from redesigning the visual encodings, data extraction also facilitates the redesign of interactions in visualizations. Using data extracted with the ReVision system [236] introduced above, Kong et al. crowdsource reference relations between visual objects in visualizations and textual descriptions to support visual linking and the highlighting interaction [154]. Transmogrification introduced above not only supports deformation-based visualization redesign but also supports adding deformation-based interactions, such as Table Lens, to existing visualizations [41]. Interaction+ is a toolkit for adding interaction functions to existing SVG-based visualizations [177]. It automatically binds interaction functions, including selection, aggregation, arrangement, comparison, filtering, and annotation, to aid the data exploration in the visualizations that may initially be static. In addition to redesigning the color mapping introduced above, using the extracted color mapping, Poco et al. provide an interactive overlay that supports cross-filtering and highlighting between the color legend and the visualization [216]. Historical visualizations are usually paper-based. Thus, most of them do not support interactions. Contemporary interaction techniques may better communicate the historical datasets than the historical visualizations.

Usage Pattern Analysis

Data extracted from a large corpus of visualizations can be used to analyze the patterns of visualization usage and derive insights. Lee et al. analyze usage patterns in millions of figures in scientific publications from PubMed [162]. They preprocess by segmenting figures containing multiple subfigures, classifying figure types, and associating the figures with the publications' bibliographic data. They find that the figure density and the distribution of figure types depend on the research field and are relatively stable over time. They also find that high-impact publications tend to have more illustrative figures. Battle et al. analyze visualization usage patterns on the web [21]. They crawl over 41000 SVG-based visualizations from major websites where developers publish visualizations, such as bl.ocks.org and Plotly. They find that most visualizations on the web are bar charts, line charts, scatterplots, and geographic maps. They also find that the most popular chart type varies across visualization authoring toolkits. For example, geographic maps are the most popular chart type among visualizations created with D3.js [32], while scatterplots are the most popular chart type among visualizations created with Plotly.

2.2.2 Techniques

Many valuable datasets are only available through their visualization imagery in printed or scanned media. Manually extracting a dataset from a visualization image with the pen-and-ruler approach is accurate but laborious and is often not scalable

for charts with many data objects, such as John Snow's cholera map [250]. Because of the various applications of such datasets, many techniques have been proposed for data extraction to reduce human effort compared with the manual approach. We categorize these techniques into four approaches: computerized pen-and-ruler, image processing, machine learning, and mixed-initiative.

Computerized Pen-and-Ruler

Many systems are digital versions of the manual pen-and-ruler approach. Users need to manually inform the machine of the locations of the visual objects encoding data. For example, to extract the data from line charts, GraphClick [34] and Ycasd [106] require the user to locate axes manually, specify the scales, and annotate the turning points on the line.

These systems are typically more efficient than the physical pen-and-ruler approach, saving users' time in measurements and scale transformations. Meanwhile, they are still laborious to use. The total number of user operations required for data extraction is typically at the same scale as the number of digitized data points.

Image Processing

Image processing techniques have been proposed for automatic *visual object detection* from statistical charts to avoid bothering users. They embed heuristics that target the visual characteristics of the visualizations to be handled. The detection of visual objects is a *prerequisite* for data extraction from visual attributes of the recognized visual objects. In the following, we categorize the image processing techniques in the related work into three approaches: connected component analysis, line and curve fitting, and pixel tracking. A work may combine multiple image processing approaches.

- **Connected component analysis**: A visual object is often a large blob of pixels with a uniform color (e.g., a solid rectangular bar in a bar chart). It is thus common to use connected component detection to identify visual objects or filter noise. Huang et al. eliminate noise in 3D pie chart images with connected component detection and filtering of small components [129]. Kataria et al. extract data from line charts where data points are marked with symbols [141]. They use median filtering to remove the lines, connected component detection to find candidate symbols and simulated annealing to segment overlapping symbols. Savva et al. use bilateral filtering to smooth the image and connected component detection with a manually set tolerance threshold to identify candidate visual objects [236]. The detected components are then filtered heuristically based on their similarity to rectangle, size, and color. The filtered components are used for data extraction from bar charts and pie charts. Gao et al. use non-parametric clustering of colors to reduce variations in pixel colors [100]. They

then use connected component detection to segment visual objects in bar charts, pie charts, and line charts.

- **Line and curve fitting**: Line detection and curve fitting methods can be utilized to identify curve objects or boundaries of visual objects appearing as pixel blobs. Zhou and Tan use Hough transform to detect parallel lines when vectorizing bar chart images [301]. Huang et al. extract line segments from binary edge images and fit elliptic arcs with the line segments to extract data from 3D pie charts with perspective distortion [129]. Liu et al. propose a general-purpose algorithm to vectorize visual objects [175]. They build connected pixel chains with run-lengths and classify each chain into a line/arc/polyline by fitting the ellipse and checking the ellipse shape and fitting errors.

- **Pixel tracking**: An alternative to the line and curve fitting is to track the pixels belonging to curve objects, typically for data extraction from line charts. Nair et al. propose a pipeline to extract data from line charts in scanned documents [191]. They compute HOG and LBP features for images in documents and train SVM to distinguish line charts for the rest images. For the line chart images, they extract axes with Hough transform, extract legend by text detection, and track lines following the color and direction of motion computed from slope gradients. DataThief [262] and UnGraph [28] are commercial systems for line chart data extraction. They track line pixels by the user-provided starting point, ending point, and line color.

After visual object detection, to extract the final dataset, additional processing is needed to decode data from the visual attributes of the visual objects. For example, to extract data from bars recognized from bar charts, it is necessary to infer the chart orientation and measure the height of the bars. Such processing is typically conducted with heuristics. In the bar chart example, assuming the bar chart is either horizontal or vertical, one may adopt Savva et al.'s strategy to build a histogram of bars' widths and a histogram of heights [236]. With the observation that bars in vertical/horizontal bar charts have the same width/height, the histogram with the larger mode implies the bar chart direction.

In addition to statistical charts, image processing is extensively used on geographic maps to extract visual objects, i.e., map symbols. Yamada et al. propose using morphological operations to extract linear and point feature corresponding to hatched areas, roads, and characters from topographic maps [283]. Miao et al. propose a generalized Hough transform method to extract symbols in topographic maps [186]. Various heuristic image processing pipelines are developed to extract different types of symbols from maps. Examples include Gamba and Mecocci's method to extract chains of symbols [99], Kerle and de Leeuw's method to extract dot symbols [145],

and Pezeshk and Tutwiler's approach to segment text symbols from the map through morphological operations [207].

The heuristic image processing techniques are suitable for data extraction from computer-generated standard statistical charts, which is the typical application scenario in the literature. These techniques are based on the known visual characteristics of particular types of charts. They inevitably set various assumptions on the visual characteristics of visualizations. They fail when the assumptions are not satisfied. Some assume a clean background, or the background noise can be easily differentiated from the visual objects by color thresholding or eliminated with algorithms such as connected component detection and filtering. Others require the visual objects in the visualizations to be standard shapes with straight or elliptic boundaries so that Hough transform or curve fitting recovers the boundaries accurately.

Thus, they are not flexible enough to handle historical visualization images, typically hindered by the non-standard visual design, the distortion of the hand-drawn shapes, and the noise due to the deterioration of the storage media, usually paper. In practice, it may thus be necessary to enable the user to tune a small number of understandable parameters to ensure a good match between the image processing techniques and minor variations of the charts. Human involvement gives rise to the mixed-initiative approach introduced in the following.

In Chapter 3, we also utilize image processing algorithms for preprocessing. Meanwhile, we do not assume the algorithms themselves are sufficient for reliable data extraction. We incorporate user intervention to handle cases where automatic algorithms fail.

Machine Learning

Designing heuristic image processing algorithms is challenging, and the heuristics are prone to errors. To avoid the handcrafting heuristic algorithms, ML has been used to construct algorithmic models for data extraction. Handcrafted algorithms have hard-coded parameter values by developers, while ML models learn parameter values from human-annotated datasets. ML approaches are commonly used to classify chart types and visual objects' roles in the visualizations.

- **Chart type classification**: Automatic data extraction techniques sometimes involve a step of chart type classification so that later processings only need to focus on one chart type at a time. For example, prior to identifying visual objects, Savva et al. classify images into 10 chart types using SVM trained on LBP and HOG features [236] Similarly, Gao et al. use SVM trained on geometric features of visual objects to classify images into 3 chart types [100]. Poco and Heer fine-tune a pre-trained AlexNet with annotated chart images to classify images into 10 chart types [215]. Chart type classification is only helpful in scenarios that

target fully automatic processing of large visualization image collections. Chart type classification models make mistakes from time to time, and errors may significantly influence the following processing. Thus, a data extraction system may better have a human in the loop to determine the chart type. It may take the user only a few seconds to make such a judgment for each image, which may avoid the error propagation of machine judgment in many cases.

- **Visual object classification**: Classification models can be trained to filter visual objects that encode data from irrelevant image elements and to differentiate the role of visual objects. Huang and Tan classify text elements by roles (e.g., title, data label, and legend label) and bound them to graphical elements in bar charts, pie charts, and line charts [128]. They use a decision tree trained with C4.5 on handcrafted features such as distance to the nearest graphical symbol and whether the text content is a number. Siegel et al. formulate line extraction from line chart images as a path-finding problem [247]. They solve the path-finding problem by optimizing an energy function with CNN. Al-Zaidy and Gile use a decision tree and a random forest trained on a bag of features to classify the roles of graphical elements and textual elements in the chart images for extracting data from bar charts [291]. Poco and Heer used SVM trained with a bag of features, such as position and aspect ratio, to classify roles of textual elements in visualization images [215]. Using the text role and content, they recover a specification of the chart. Our approach to data extraction from historical visualizations also involves classifying visual objects, as introduced in Section 3.4.

The combination of machine learning and image processing presents an attractive advantage when there are many annotated visualization images available for training a reasonably accurate model and many more not-yet-annotated visualization images for which the trained model can be used to recover the unknown data. Such an advantage is absent with historical visualization images since the number of similar images available usually is insufficient as the training data. By the time all the similar visualization images are annotated for model training, there is no need for the model anymore.

This challenge leads to our solution: one may select parts of an image to train a model with possibly mediocre accuracy and apply the trained model for the rest of the image and, if any, similar images. This idea is the basis for Chapter 3. While all the previous techniques train models offline, we formulate data extraction as an iterative machine learning process, as introduced in Section 3.3. We further enhance this idea by introducing an algorithmic selection of “parts of an image” and an algorithmic suggestion of “tentative labels” to reduce the number of user operations required for interactive classification. We evaluate the merits of these two additions in Section 3.5.

Mixed-Initiative

In the above, we have introduced various existing approaches to data extraction, including manual measurements, image processing algorithms, and pre-trained ML models. These approaches either overly rely on the user, which can be accurate but laborious, or overly rely on algorithm processes, which can be efficient but inflexible or inaccurate (see Section 3.2 for an extended discussion).

Semi-automatic techniques thus have been used in some systems to balance data extraction accuracy and user burden. These systems support the interleaving of contributions by algorithms and user intervention. They are categorized into mixed-initiative systems under the definition by Horvitz [124] as detailed in Section 2.3.1.

Yang et al. propose a semi-automatic system for data extraction from line charts, bar charts, and pie charts [284]. They break down data extraction into the steps of vectorization, text extraction, and chart component extraction. Each step is first conducted by image processing or heuristic rules and then allows the user to manually correct algorithm mistakes, such as editing the vectors' control points.

WebPlotDigitizer supports data extraction for bar charts, line charts, polar charts, and ternary diagrams, as well as distance measurement for maps and microscope images [229]. Similar to Ycasd [106] and GraphClick [34], it provides a set of computerized pen-and-ruler tools, such as specifying an axis by marking two points in the chart. It also provides algorithmic functions to assist the user, such as tracking lines in line charts by color. The user can intervene by tuning algorithm parameters or manually editing the extracted data points if the algorithm output is unsatisfying.

iVoLVER enables users to extract data from multiple sources, transform the data, and construct new visualizations with the data [185]. It uses standard image processing algorithms, such as connected component detection and morphological operations, to extract visual objects. The user then has to manually specify the extracted visual objects' attributes (e.g., position, length, or area) that maps to the data.

ChartSense enables users to determine image processing parameters [138]. It contains specialized interactive image processing pipelines for 5 types of charts. The user can specify critical chart features and textual information to be used in the data extraction process.

Through these examples, we observe three types of user interventions:

- **Tuning algorithm parameters**: The system may support the user to tune the algorithm parameters to run algorithms or rerun the algorithms if the previous outcome is unsatisfying. For example, the user can tune the color used for matching pixels of visual objects in WebPlotDigitizer [229]. The user more or less needs to know the algorithm to efficiently and effectively tune the parameters.

- **Correcting algorithm outputs**: The system may support the user to directly fix the machine's incorrect intermediate results or final outcomes (i.e., extracted

data). For example, the user can edit polygons created by vectorization, which are intermediate results, in Yang et al.'s system [284] and ChartSense [138]. The user can correct extracted data points, which are the final outcomes, in WebPlotDigitizer [229].

- **Providing intermediate extraction results**: The system may require the user to provide information about intermediate extraction results. For example, WebPlotDigitizer [229] requests the user to specify the location of axes in the image. iVoLVER [185] requests the user to specify the visual channel that encodes data. In these cases, the systems essentially leave the challenging subtasks of data extraction to the user.

As we aim to recover data from historical visualizations, the quality of data extraction is a crucial requirement. Hence, involving users in the extraction process is unavoidable. The approach proposed in this thesis for data extraction also falls into semi-automatic mixed-initiative techniques.

Meanwhile, the existing semi-automatic systems typically do not take the reduction of user burden as their design principle. They frequently involve laborious user operations, such as editing polygons. A user's specification of various image processing parameters may improve the accuracy of data extraction from historical visualizations. Meanwhile, the trial-and-error in tuning the parameters could easily incur undesirable effects, e.g., time cost, frustration, and cognitive load.

Our systems introduced in Chapter 3 and Chapter 4 support all three types of user interventions mentioned above. The user can tune algorithm parameters when configuring a workflow, correct algorithm outputs during quality assurance of machine predictions, and provide intermediate extraction results such as extracting visual objects by manually creating polygons. We expect quality-assuring and correcting algorithm outputs to be the most common and essential form of interactions in our systems, as it directly contributes to the final outcome of data extraction, while the other two types of interventions may require trial-and-error and can be laborious.

This thesis investigates how to better join human and machine strengths in the data extraction and data labeling scenarios. Our approach differs from existing semi-automatic techniques in that we utilize machine learning techniques to reduce user burden instead of heuristic image processing algorithms. Chapter 3 introduces our approach to convert data extraction into classification tasks and integrate machine assistance to reduce user burden. With our approach, the user intervenes by answering questions raised by the system to correct algorithm outputs. Answering the questions does not require knowledge of the underlying algorithms or mapping specifications. Meanwhile, our system can reduce the users' burden in answering the questions by improving the correctness of the default answers using active learning.

2.3 Interactive Machine Learning

This thesis proposes an interactive machine learning approach to data extraction and data labeling. The section starts with an introduction of related *concepts* to provide a context. Then, we introduce interactive machine learning *systems* with data labeling functions. The section ends with a brief review of weakly-supervised learning *algorithms* that may reduce user effort in interactive machine learning and specifically focus on active learning and semi-supervised learning algorithms that we have utilized in our systems.

2.3.1 Concepts

This section clarifies concepts related to the thesis, including interactive machine learning, intelligent user interface, mixed-initiative, and visual analytics.

Interactive Machine Learning

Interactive machine learning (IML) systems refer to *interactive systems that support user intervention in the iterative machine learning process to accomplish tasks jointly or improve the model training*. Compared with conventional ML processes, IML techniques may help domain experts with no professional programming and ML expertise to utilize ML techniques. This thesis falls into the category of IML research, as we also utilize user inputs to train models incrementally.

In some cases, IML is defined narrowly to focus on model training, with the trained model being the output. For example, Dudley and Kristensson define IML as "an interaction paradigm in which a user or user group iteratively builds and refines a mathematical model to describe a concept through iterative cycles of input and review" [78]. The narrow definition may originate from the early research effort on IML that focus on model training. Ankerst et al. propose an approach for interactively training decision trees that enable users to select split points in the dimensions corresponding to decision nodes in a circle segment visualization [16]. Ankerst et al.'s work is one of the earliest works on IML. Building on Ankerst et al.'s work, Ware et al. enable interactive training of decision trees where the decision criteria can be based on pairs of attributes as visualized in scatterplots [271]. Ware et al.'s work is the earliest work we found that introduces the concept of IML in an HCI venue.

In many other cases, IML is defined broadly to cover processes where user intervention is used for iterative model training, and the output or goal does not need to be the trained model. For example, Ramos et al. defined the IML process as "one in which a person (or people) engages with a learning algorithm, in an interactive loop, to generate useful artifacts" where the artifacts can be data, insights about data, or machine-learned models [222]. Similarly, Jiang et al. define IML as "an iterative

learning process that tightly couples a human with a machine learner" to accomplish various tasks such as data analysis, interactive labeling, and information retrieval [131]. Under the narrow definition, this thesis is related to IML. Under the broad definition, the approach in this thesis belongs to IML. Throughout the thesis, we refer to IML as its broad definition. Section 2.3.2 reviews the systems and techniques related to IML.

Intelligent User Interface

Intelligent user interface (IUI) is an interdisciplinary field that studies *interactive systems that incorporate some form of machine intelligence*. It is a broader concept than interactive machine learning. The interactive systems proposed in this thesis utilize machine intelligence to assist the user in data extraction and data labeling and thus belong to IUIs.

A synonym of IUI is the "interactive intelligent system" (IIS). The similarity of the two concepts can be seen through the definitions in the corresponding conference and journal. IUI is defined as "systems that people directly interact with and that incorporate some aspects of machine intelligence" by ACM International Conference on Intelligent User Interfaces[1]. Similarly, IIS is defined as "interactive systems that incorporate certain form of machine intelligence" by ACM Transactions on Interactive Intelligent Systems[2].

Note that in software engineering, the term "interface" sometimes refers to only the front end of an interactive system and excludes the back-end algorithms and databases. Although the term "intelligent user interface" ends with "interface", its scope is well beyond the front end and, in fact, generally covers all parts of an interactive system.

In addition to IML, IUI covers various topics, such as adaptive user interfaces, user modeling, and conversational interfaces [117]. The following gives a brief impression of IUI research through examples.

- **Adaptive user interface**: The interface may self-adapt its design parameters according to various sources of information, such as interface specification, user interactions, and the display device. The SUPPLE system adapts the interface rendition to minimize user effort according to the rendering device property and user operations cost recorded in usage traces [98]. Todi et al. use reinforcement learning to schedule the sequences of adaption in menu design when minimizing pointing time costs [259]. PLIANT [102] and PTIME [273] systems use active learning to adapt the model parameters for calendar scheduling recommendations. These systems also fall under the broad definition of IML systems.

- **User modeling**: The user interface may explicitly model the user to facilitate "intelligent" interactions [272]. For example, Dabek and Jesus use non-deterministic

[1] https://iui.acm.org/2022/call_for_papers.html
[2] https://dl.acm.org/journal/tiis/about

finite automaton to model user interaction sequences and recommend interactions for future users [71].

- **Conversational interface**: A classic application scenario of IUI is conversational interfaces. Nguyen and Wobcke propose a conversational interface for email management based on an agent architecture with belief, desire, and intention attitudes [195]. The Iris system translates nested conversations with the user into python functions for data science applications [87]. In conversational interfaces, a critical component is the turn-taking mechanism for mixed-initiative interaction, which is introduced below.

A feature characteristic of IUI systems is adaptiveness, as summarized by Edmonds [80] and Völkel et al. [267]. These systems may adapt their interface design and algorithmic parameters according to the scenario and user input, as is prominent in adaptive user interfaces and interactive machine learning systems.

These systems join human and machine strengths. They are critical for scenarios where purely relying on human intelligence is laborious and costly, while purely resorting to machine intelligence leads to imprecise and untrustworthy results. The fundamental goal of integrating machine intelligence in IUI systems is not to fully automate the task. If the task can be fully automated at the beginning, it might have been a waste of developers' effort to build an interface and users' effort to interact with it. Instead, the goal is to enable the user, e.g., to reduce errors and promote productivity [80], while making the experience friendly and natural for the user [183].

Mixed-initiative

Mixed-initiative systems support the *interleaving of contributions by users and machines*. It is a classic idea that dates back to the 1970s. The earliest reference we have found is Carbonell's system, which uses a graph of facts, concepts, and procedures to support mixed-initiative tutoring dialogues between the machine and students [46]. Later, in the 1990s, as a response to the debates on direct manipulation interfaces versus interface agents [246], Horvitz suggests that mixed-initiative systems may join the advantages of both sides [124]. The research in this thesis falls into this category of mixed-initiative systems. Our systems for data extraction and data labeling in Chapter 3 and Chapter 4 involve user intervention to contribute data labels and the machine to learn from the user to improve the automation of the tasks gradually.

The definition we adopt above originates from Horvitz's broad definition of "mixed-initiative" as "methods that explicitly support an efficient, natural interleaving of contributions by users and automated services aimed at converging on solutions to problems" [124]. This definition reveals a fundamental idea shared by mixed-initiative systems, including ours, that although many tasks are hard to automate, they can benefit from machine assistance.

In some other cases, the term "mixed-initiative" narrowly refers to systems that support "automated control of turn-taking" to enable agents, including humans and machines, to take initiative flexibly [8]. Examples include conversational systems where the machine may not wait for the user's response but may take the initiative to start a new topic. The work in this thesis does not fall into this narrow definition of "mixed-initiative" systems. In our systems, user and machine initiatives are predefined by workflows such as the one in Fig. 3.3.

Visual Analytics

Visual analytics (VA) is a subfield of data visualization. It is defined as "the science of analytical reasoning facilitated by interactive visual interfaces" in Thomas and Cook's *"Illuminating the Path"* [256]. The goal is to support human judgment when analyzing large amounts of information in mission-critical applications, such as protecting national security against terrorism.

Later, Keim et al. propose a more detailed definition: "visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex data sets" [142]. They propose a VA model that integrates human visual data exploration in visualizations and machine data analysis algorithms. Through iterative interactions with the machine, the human gains knowledge of the data and drives the machine to be more focused.

From these definitions, it can be summarized that VA supports human decision-making and knowledge generation from massive complex datasets. It features machine assistance to augment human information processing capacity and interactive interfaces where the user iteratively communicates with the machine.

Our approach to data extraction and data labeling is similar to the original idea of VA. Both combine the strengths of humans and machines through iterative interactions because of the sheer amount of data to be processed. Meanwhile, our goal is different. VA's goal is typically to help users gather high-level knowledge or make complex decisions. By comparison, our goal is to gather structured data from the raw input data, and whether the user gains insight from the process is not critical. A common challenge for VA applications is that user tasks are vaguely defined, thus making evaluation difficult. For our scenario, the tasks are well-defined, allowing us to carry out a model-based evaluation in Chapter 5.

2.3.2 Systems

IML involves user interventions in iterative machine learning processes. It has been applied to various applications, such as medical diagnosis [44], quality control in manufacturing [81], image restoration [274], and audio source separation [42].

The goal of IML processes may be the trained model to be reused and may also be model output used for solving domain-specific tasks [11]. This thesis focuses on the latter scenario. Our goal is to get the extracted data and labeled data while the trained models are disposable side effects.

In IML processes, the user may intervene various steps, such as feature selection [9], model tuning [55], testing [56], and explanation [252]. In quality-sensitive applications, quality assurance of model predictions and error identification is important [146]. Among the various steps the user may intervene, a common form of user involvement is to label data for iterative model training [78]. For example, IML systems for information retrieval may utilize relevance feedback explicitly or implicitly labeled by the user to iteratively improve the relevance of the retrieved result [127].

This thesis concerns the scenarios of data extraction from historical visualizations and data labeling. Data extraction can be seen as an instance of data labeling tasks. The process of associating a dataset with a visualization image can be seen as providing a structured label, the extracted dataset, for a visualization image. Chapter 3 reduces this structured labeling task to a series of simpler labeling subtasks. In the following, we review IML systems that enable users to provide data labels. Specifically, we focus on the systems that provide machine assistance in the user's labeling process to improve efficiency. We categorize their focuses into semi-automatic labeling, active learning, cluster labeling, and task design. We also introduce relevant frameworks and toolkits in interactive machine learning.

Semi-Automatic Labeling

To reduce user burden, a common strategy is for the system to support semi-automatic labeling by learning from the user inputs and providing default labels. This way, the user only needs to quality-assure the model predictions and correct machine mistakes, which typically takes less time than labeling from scratch. In the context of data extraction from historical visualizations and data labeling, quality assurance ensures the correctness of the extracted dataset. This approach dates at least back to the 1990s when Marcus et al. used semi-automatic labeling to build an English treebank for NLP [182].

Typically, the default labels to be quality-assured in semi-automatic labeling are provided by machine-learned models. Fluid Annotation [15] uses a pre-trained neural network to propose a set of possibly-overlapping segments to help annotate image segmentation. The user can then edit the geometry of the segments to correct errors. V-Awake [101] focuses on time series segmentation. It uses LSTM to assign tentative labels and visualizes the model information to help annotators diagnose and correct model predictions.

In some cases, the default labels to be quality-assured may also come from crowdworkers. LabelInspect [176] is a visual analytics system for improving crowdsourced

image labels. It enables the user to improve the quality of crowdsourced labels by identifying uncertain instance labels and unreliable crowdworkers. DataDebugger [280] is a system for image label correction based on the idea of data debugging using trusted items. It allows the user to identify data objects with suspicious labels and correct the labels, specify data labels to be trusted, and propagate the trusted labels.

When the default labels come from machine predictions, the machine assistance may be gradually improved by incrementally learning from user inputs to reduce users' quality assurance effort. Fails and Olsen propose Crayons [84] for image pixel classification. The user can paint pixels to correct machine predictions and provide training data to the machine. It learns the labels generated from the user's paint interaction. Fogarty et al. develop a search system, CueFlik [91], with which the user can define search criteria for a concept by using positive and negative examples and rank search results according to the similarity to the concept. ISSE [42] algorithmically suggests refined segmentation to help annotators separate sound into its respective sources by painting on time-frequency visualizations.

In addition to reducing user burden, semi-automatic labeling may also influence the label quality positively or negatively. Semi-automatic labeling may improve label quality in subjective tasks, as Green et al. [105] observe in annotating language translation. Meanwhile, semi-automatic labeling may introduce the default effect [137] where annotators are biased towards accepting defaults.

Chapter 3 and Chapter 4 utilize the semi-automatic labeling approach to reduce user burden. Chapter 5 proposes using model-based evaluation for quality assurance interfaces in semi-automatic labeling.

Active Learning

Another family of techniques commonly used in labeling systems is active learning which focuses on prioritizing user effort. Active learning methods attempt to select more informative data objects and request that users label them interactively. This way, the user is guided to provide examples that help efficiently improve machine assistance.

Active learning has been used in IML systems for various applications, such as document classification, image classification, and recommendation systems. Settles propose a document annotation system that incorporates active learning for selecting documents to be labeled and learns from both document and keyword labels from the user [242]. Heimerl et al. presented a visual analytics system for text document classification based on active learning [118]. They evaluated three interface variants for building classifiers with different levels of user control and visualization integration. Kucher et al. propose ALVA, a technique for multi-label document annotation and model training in sentence stance detection [156]. They demonstrate a labeling system with an active learning function for data annotators and a visual analytics system for data analysts to analyze the labeling and training process. To support efficient

iterative image searching, Luo et al. proposed multi-class query ranking [178], a semi-supervised learning algorithm based on manifold ranking. Gervasio et al. [102] and Weber and Pollack [273] used active learning techniques to learn users' scheduling preferences. Their method balances learning efficiency and user satisfaction.

Chapter 3 and Chapter 4 utilize active learning in determining the order for data objects to be labeled. The examples introduced above focus on systems that incorporate active learning. Section 2.3.3 reviews active learning algorithms.

While, conventionally, active learning focuses on algorithmic selection strategies, the selection may also involve user intervention. For example, projection scatterplots with iso-contours visualizing label uncertainty have been used to assist users in data selection [169]. Seifert and Granitzer proposed a system for image labeling that allows the user to actively select data objects to label with the aid of visualization [240]. Our toolkit in Chapter 4 provides projection scatterplots as an interface module to determine the order for data objects to be labeled.

Cluster Labeling

A data labeling interface may present the user with a batch of data objects simultaneously for the user to assign labels batch by batch instead of one by one. Clustering is frequently used in data labeling systems to support batch labeling. It groups similar data objects that likely share the same label so that the user may label multiple data objects with one click [253]. It thus may improve the efficiency of human inventions.

Cui et al. described a photo annotation system, EasyAlbum [67], allowing the user to annotate data objects in a cluster-by-cluster manner. Liu et al. described a method that first divides unlabeled data objects into clusters, selects exemplars from each cluster for the user to label, and propagates the labels to other data objects in the cluster [174]. Rafailidis et al. described the content-based tag propagation technique that propagates user-provided tags to similar items to address the "cold start" problem and boosts the accuracy of tag-based search engine [220]. Tian et al. described a hybrid method that first groups unlabeled data objects into evident clusters and a background cluster. It then allows the user to annotate each evident cluster as a whole and guides the user to annotate the background cluster using an active-learning-based method [257]. Tang et al. described a multi-scale method that allows the user to label image segments in a cluster-by-cluster manner and to refine each cluster recursively [255]. Out toolkit in Chapter 4 provides clustering as an algorithm module to group similar data objects.

Task Design

Another branch of study focuses on designing the presentation of tasks to the user. A prominent example is gaming with a purpose (GWAP) [5] in which casual users are instructed to play games to contribute data labels implicitly. Techniques in this

category, such as ESP game [4] and Peekaboom [6], focus on the gameplay design that strives to motivate causal users. Similarly, II-20 supports the categorization of image collections and features a gameplay design following Tetris metaphor [290].

Another example is embedding labeling tasks in human/bot verification [7, 285]. A large body of work on crowdsourcing annotations relates to task design, which typically focuses on task assignment schemes to collect high-quality labels with low cost and latency [51].

Frameworks and Toolkits

The examples above show common themes in interactive machine learning systems, such as using semi-automatic labeling and active learning to save users' effort. These techniques are essentially building blocks that can be composed into workflows according to specific design patterns. The following introduces the literature proposing design patterns in IML systems, typically in the form of generic workflows composed of conceptual modules.

Höferlin et al. presented an active learning workflow for training classifiers in the context of video analytics [121]. Their active learning workflow contains three stages: (1) algorithmic sampling with an option for user sampling with the aid of visual analytics, (2) manual annotation, and (3) training classifier with an option for manual modification of a classifier with the aid of visual analytics.

Bernard et al. proposed an active learning workflow for learning a similarity metric in sports data analysis [25]. It includes three components: (1) a visual-interactive learning interface, (2) similarity modeling by learning weights of features, and (3) a list-based result visualization that ranks data objects by the learned distance.

Building on the two case studies [25, 121], Bernard et al. outlined a conceptual framework for visual interactive labeling [26], which features six components: (1) preprocessing and feature extraction, (2) learning model, (3) result visualization, (4) candidate suggestion, (5) labeling interface, and (6) feedback interpretation. This framework advocates the close integration between interactive visualization with active learning to enable concurrent objectives, including data labeling, model training, and knowledge acquisition by the user.

Other frameworks related to interactive machine learning include:

- The conventional active learning framework summarized by Settles following a survey of the active learning literature [241]. It includes three main stages: (1) algorithmic sampling, (2) interactive annotation, and (3) classifier training.

- The human-centered ML framework by Sacha et al. [235], which includes (1) data editing and enrichment, (2) data preparation, (3) model selection and building, (4) exploration and direct manipulation, and (5) execution and evaluation.

- The VIS4ML ontology by Sacha et al. [234], which divides ML workflows into four stages: (1) prepare data, (2) prepare learning, (3) model learning, and (4) evaluate model. They further list 24 processes in ML that can benefit from interactive visualization.

- The visual classification workflow by Paiva et al. [204] that concerns (1) sample selection, (2) manual labeling, (3) model training, and (4) label verification.

- The general pathway from data mining to knowledge discovery by Fayyad et al. [88], which includes (1) goal identification, (2) dataset creation, (3) cleaning and reduction, (4) matching KDD goal with a data mining method, (5) exploratory analysis, (6) data mining, (7) result interpretation, and (8) action based on the discovered knowledge.

- Various frameworks for visualization and interaction, e.g., the operator interaction framework by Chi and Riedl [61], the data state reference model for visualization by Chi [60] and interactive visualization workflows by Card et al. [47] and van Wijk [275].

Similar to the line of work, this thesis also focuses on developing conceptual frameworks. Chapter 3 proposes a generic workflow for data extraction from historical visualizations. Our workflow involves various modules, including algorithmic sampling, algorithmic default labeling, interactive labeling, training classifier, stoppage analysis, and quality assurance. Our approach falls into the conceptual frameworks of Bernard's [26] and Sacha's [234, 235]. Chapter 3 focuses on data extraction applications In our approach, every decision made by users is primarily for the objective of extracting data as quickly as possible and with as few user operations as possible. The effort for the users to "teach" an ML model by correcting its errors is to enable the model to provide helpful assistance to the users as soon as possible. The users make simple decisions to correct the model's errors and make complex decisions about the right moment to stop "teaching" the model.

The commonalities in the workflows in the literature lay the foundation for Section 4.3 where we summarize shared conceptual modules across interactive machine learning systems for data labeling. Similar to the literature on generic workflows, we aim to summarize design patterns in data labeling systems. Instead of a single workflow, our solution is a generative framework for building workflows. We believe that a single workflow cannot be optimal for all application scenarios of data labeling. Meanwhile, our framework builds on existing workflows, as they are included in the corpus for summarizing common modules through coding (in Section 4.3).

Based on the common modules identified in Section 4.3, we develop an authoring toolkit in Chapter 4 that can be used to build data extraction and data labeling systems. The work introduced in Chapter 4 falls into the family of toolkit research in HCI [161]

that contributes techniques for building new systems. Examples in this body of toolkit research are the ones that assist the development of visual interfaces, such as D3 [32], iVisDesigner [226], and iVoLVER [185].

Meanwhile, to the best of our knowledge, there is no prior work on extensible systems for building data labeling systems. A relevant thread of work that aims to save the cost of developing labeling systems is data labeling platforms and systems that support multiple labeling applications. Examples include the project templates of Amazon Mechanical Turk [10] and Prodigy [83]. While the template-based approach can address some common use cases, extending to new application scenarios requires implementing new templates from scratch, and reusing software modules is not supported. LabelStudio [258] also provides a collection of templates with more flexibility. While it allows the user to use active learning and pre-labeling in the labeling interface, adding other algorithm modules and customizing interface modules in LabelStudio is difficult.

The toolkit we propose in Chapter 4 also aims to promote reuse like these systems but focuses on reuse at the module level instead of the template level. Additionally, building on the conceptual framework introduced in Section 4.3, OneLabeler supports a more extensive coverage of the design space of data labeling techniques.

Marcelle is a library that enables the user to compose interactive machine learning workflows with various components [92]. Marcelle and Chapter 4's toolkit are similar in that both attempt to modularize reusable components in interactive machine learning. They are different in that our toolkit focuses on data labeling scenarios and supports the visual configuration of higher-level reusable components (e.g., interactive selection within a projection). Marcelle aims for general-purpose interactive machine learning systems and requires textual programming of lower-level reusable components (e.g., UI widgets).

2.3.3 Algorithms

In interactive machine learning systems, while user interventions are necessary, avoiding too much burden on the user is desirable. Weakly supervised learning techniques focus on learning from weak supervision, such as *incomplete, inexact, and inaccurate supervision*.

We suggest that *weakly supervised learning techniques are generally helpful in improving user productivity in data labeling*. With weakly supervised learning techniques, the IML system may be able to learn from incomplete, inexact, or inaccurate labels, which may be less laborious for the user to provide compared with complete and fine-grained labels.

This section introduces weakly supervised learning algorithms and focuses on active learning and semi-supervised learning that we have utilized in Chapter 3 and 4.

According to Zhou's categorization of weakly supervised learning techniques, active learning and semi-supervised learning can be regarded as learning from incomplete supervision where only a part of the dataset is labeled [302].

Active Learning

Active learning is a family of ML methods interacting with the user during a learning process. In a narrow definition, it is for seeking labels for unlabeled training data objects to improve the performance of the ML process. In a broad definition, it is for seeking important decisions that can improve the quality and performance of the ML process. The decisions are not limited to deciding labels of data objects but can also be choosing features [242] and deciding when to stop the learning process [29]. Its underlying idea is to let users focus on annotating important data when annotating all the unlabeled data is undesirable or infeasible.

A critical feature of many active learning methods is to iteratively select data objects "intelligently" to seek labels from the user. Various selection strategies have been proposed in the literature. The following categories selection strategies as serial or batch-mode by whether one or multiple data objects are selected each time, according to Settles' terminology [241].

- **Serial**: Lewis and Catlett propose estimating the uncertainty of each unlabeled data object and selecting the one with the highest uncertainty for the user to label [167]. Uncertain data objects are informative and may help reduce the learned model's uncertainty. Nguyen and Smeulders use the clustering property of the data to select data objects in favor of dense clusters [196]. These data objects may represent the cluster patterns and may help the model effectively capture the patterns. Lin et al. propose the re-active learning scheme that allows the algorithm to sample both labeled and unlabeled data objects for the user to label or verify [171]. In general, serial selection strategies score unlabeled data objects by criteria in favor of informativeness and representativeness and choose the one with the highest score for the user to label.

- **Batch-Mode**: While active learning algorithms mainly focus on the serial selection that selects data objects one at a time, some consider the scenario of batch-mode selection that selects multiple data objects simultaneously. A scenario where batch-mode selection is necessary is when the interface presents a batch of data objects each time to support batch labeling, as we adopt in Chapter 3. Although it is feasible to directly adapt serial methods and select a batch of data objects with the highest scores, this approach may be suboptimal as it ignores the information overlap between the data objects [241]. Brinker introduces an active learning method based on diversity to sample a batch of

unlabeled data objects for the user to label [38]. Xu et al. describe a batch method that selects data objects for labeling based on their relevance, density, and diversity measures [281]. Guo and Schuurmans describe a batch method that selects the data objects that maximize the discriminative performance of the target classifier while minimizing the entropy of the missing labels [109]. While the active learning strategy can be machine-centric, focusing on optimizing machine performance, it may also be designed to optimize user time cost. Deng et al. propose a strategy to select data objects to label that maximize annotation's utility-to-cost ratio in multi-label classification [72]. Kveton and Berkovsky proposed a generalized linear search for providing users with batch recommendations to reduce the interaction cost for search activities [159].

Because of active learning's focus on the iterative learning process with the user in the loop, it is frequently used in interactive machine learning systems. As introduced in Section 2.3.2, such examples include Gervasio's [102] and Weber and Pollack's calendar management system [273], Heimerl et al'.s text classification system [118], and Liao et al.'s video labeling system [169].

The data extraction and data labeling systems introduced in Chapter 3 and Chapter 4 utilize active learning to determine the order for representing data objects. In particular, we have used the methods by Lewis and Catlett [167], Brinker [38], and Xu et al. [281].

Semi-Supervised Learning

Semi-supervised learning is a family of ML methods using both labeled and unlabeled data objects to train a model. Its underlying assumption is that the data distribution $p(x)$ provides information about the posterior distribution $p(y|x)$. Based on Zhu's categorization [304], the following groups semi-supervised learning techniques into three approaches: graph-based, low-density separation, and adaptors of supervised models.

- **Graph-Based**: Graph-based methods build graphs with labeled and unlabeled data objects being nodes, and the similarity between data objects reflects on the edges. Zhu and Ghahramani introduce a graph-based algorithm for transferring known labels to unlabeled data objects iteratively [305]. Zhou et al. propose another graph-based algorithm with a different iterative label spreading function inspired by spreading activation networks [300]. Tong and Jin describe a mixed label propagation algorithm [260] that exploits the information of similarity and dissimilarity among data objects. Chapelle et al. proposed a framework for incorporating unlabeled data in kernel classifiers [53]. The kernels are derived from the graph Laplacian matrix.

- **Low-Density separation**: Low-density separation methods follow the rationale that the classification boundary should not go through regions with high data density. It is essentially the same idea as conventional SVM's idea of margin maximization. Following this idea, Joachims develops the transductive support vector machine as an extension of the traditional SVM by involving unlabeled data in addition to labeled data for margin maximization [132]. Leistner et al. extend random forests with a semi-supervised mechanism [163]. They show that conventional decision trees optimize multi-class margins and reformulate the optimization goal to utilize unlabeled data.

- **Adaptors of supervised models**: Aside from the above models designed deliberately for semi-supervised learning, adaptors have been proposed to convert existing supervised models for semi-supervised learning. The adaptors are usually agnostic of the adapted model. Yarowsky proposed self-training [286], a boosting technique that uses pseudo-labels of unlabeled data during training. After using a supervised classifier trained on labeled data as an interim ML model, the technique applies the interim model to unlabeled data and extends the training dataset used to retrain the model. Blum and Mitchell proposed co-training [30] that uses two interim classifiers trained using two disjoint subsets of the data features. The predicted tentative labels on the unlabeled dataset from one classifier are fed to the other to enlarge the training set.

The data extraction and data labeling systems introduced in Chapter 3 and Chapter 4 utilize semi-supervised learning algorithms to propagate labels to unlabeled data objects. In particular, we designed a graph-based label propagation process based on Zhou et al.'s method [300] as a default labeling component in our implementation.

Others

Aside from active learning and semi-supervised learning that targets the challenge of incomplete labels, other weakly supervised learning techniques have been proposed to address the challenges of inexact and inaccurate supervision.

- **Learning from inexact supervision**: In the case of inexact supervision, the training data only give coarse supervision. In interactive machine learning, allowing the user to provide inexact supervision may reduce effort compared with providing fine-grained supervision. A scenario is learning to segment images with only image classification labels provided in the supervision. Multiple instance learning techniques work with inexact supervision that assumes a set of labeled bags are available, with each bag containing multiple instances [74]. In the binary classification scenario, a bag is labeled negative when and only when all the instances in the bag are negative. Wu et al. use multiple instance learning

for interactive image segmentation [277]. They suggest that each horizontal and vertical sweeping line in the user-provided bounding box should contain at least one foreground pixel and thus converts to a multiple instance learning problem with lines being bags and pixels being instances.

- **Learning from inaccurate supervision**: In various cases, data labels contain noise, referred to as inaccurate supervision. It is typically the case for crowdsourced labels. Brodley and Friedl propose using model ensembles to identify mislabeled data objects [40]. They learn an ensemble of models using different ML algorithms from training data subsets and identify outlying data objects with the models. Zhu et al. propose a technique to identify data objects with noisy labels in large datasets [306]. It partitions the dataset into subsets, learns classification rules for each subset, selects good rules, and evaluates whether a data object is mislabeled by checking against the good rules. Raykar et al. propose a method that jointly learns the ML model, annotator accuracy, and the estimated label from crowdsourced labels [225]. They demonstrate that their approach performs better than the commonly used majority voting for processing crowdsourced labels. Northcutt et al. propose estimating the confidence on data labels for the user to prune noisy data [200].

2.4 Usability Evaluation

Chapter 5 introduces using simulation to estimate task completion times in quality assurance interfaces. This section reviews related usability evaluation approaches. We focus on model-based evaluation methods that our approach belongs to.

Evaluation is to assess the usefulness and usability of interactive systems and interaction techniques [75]. It is helpful in various stages of iterative software engineering processes for interactive systems. During prototyping, evaluation may guide software development to improve usability. After implementation, evaluation may derive summative results on whether the interactive system achieves specific goals, such as improved efficiency over existing systems.

2.4.1 Overview

Usability evaluation methods can be categorized by two dimensions: involvement of users and methodological approach [20]. An empirical evaluation involves users (also referred to as "testing method" [14] in the literature). It can also be analytical, which requires evaluation experts to carry out (also referred to as "inspection method" [14] and "expert analysis" [75] in the literature). The evaluation methodology can be qualitative, quantitative, or combined.

Empirical evaluation captures payoff measures, such as user performance, to infer interface properties [104] with the following example methods.

- *Laboratory studies* involve a sample of users to perform tasks and analyze objective measurements, such as task completion time and accuracy. It is prevalent in HCI research nowadays.
- *Think-aloud* gains insight about the interface through the records of what the user said that came into their mind while performing tasks with the interface.
- *Questionnaires*, such as Likert scales, gather user ratings of interfaces.

Analytical evaluation inspects the interface's intrinsic properties influencing usability [104] with the following example methods.

- *Heuristic evaluation* involves a team of evaluators checking an interface against a short list of design guidelines and aggregating the identified issues. It was first formalized by Nielsen and Molich [197].
- *Cognitive walkthroughs* are a technique where evaluators mentally walk through a task following a novice user's mindset to identify usability issues [165].
- *Model-Based evaluation* involves modeling users' physical and mental operations to accomplish a task in an interface [238]. It is sometimes also called "action analysis" [166]. The modeling enables the estimation of usability metrics, such as task completion times [48] and learning time costs [147].

Analytical methods are usually carried out by evaluation experts. Analytical methods do not directly measure the performance of target users and thus may not be as conclusive as empirical methods. Meanwhile, analytical methods can be less costly than empirical methods, as the former typically involve fewer people and thus are more suited for early-stage design and prototyping iterations. Additionally, it is usually easier to attribute the causes of usability issues with analytical methods. Chapter 5's simulation approach belongs to analytical methods, specifically model-based evaluation.

2.4.2 Model-Based Evaluation

Chapter 5 uses the model-based evaluation approach for quality assurance interfaces in interactive machine learning. Generally, model-based evaluation involves modeling the user's task completion procedure in an interactive system to estimate usability metrics, such as task completion time or learning time, through simulation or calculation. Kieras identifies three categories of models in model-based evaluation: task network models, cognitive architecture models, and GOMS models [238]. We introduce examples following this categorization.

- *Task network models* capture the dependencies between user and machine tasks as networks [155]. SAINT (Systems Analysis of Integrated Networks of Tasks) is a simulation language developed for task network models [65]. It supports the estimation of task completion time distribution through Monte Carlo experiments, given the distributions of probabilistic variables in the model.

- *Cognitive architecture models* apply theories in cognitive science to model human motor control, perception, and cognition [12]. Examples include the ACT-R architecture [13] and the EPIC architecture [150]. Both of them model users' low-level interaction with production rules that depict user reactions to stimuli but differ in some aspects of their human model. ACT-R concerns serial production cycles, while EPIC concerns parallel production cycles.

- *GOMS models* describe the required procedural knowledge for a user to carry out routine tasks in user interfaces, similar to depicting computer algorithms with pseudocode [133, 134]. The term "GOMS" stands for its four components: goals, operators, methods, and selection rules. Various GOMS techniques exist, such as the Keystroke-Level Model [48]. Chapter 5's work falls into this category.

Model-based evaluation has been used in various application scenarios. The EPIC architecture has been used to predict human performance in telephone operator tasks [150]. John and Vera use GOMS to model the highly interactive task of video game playing [135]. Gond and Kieras use GOMS to identify usability issues and aid the redesign of ergonomics assessment software [103]. Beard et al. apply GOMS to evaluate the user interface for displaying computed tomography images for medical diagnoses [23]. Kieras and Santoro apply GOMS to evaluate the user interface and collaboration scheme for shipboard workstations [149]. Kieras and Hornof introduce a GOMS model for the visual search task [148]. Ramkumar et al. apply GOMS analysis to image segmentation interfaces [221]. Azzopardi et al. model and simulate the trade-off between the query and assess interactions to the total time cost in information retrieval systems [18], and validate the simulation result through a user study [19]. Like this line of work, our modeling and simulation in Chapter 5 follow the model-based evaluation approach. We focus on a different scenario of evaluating data extraction and data labeling systems.

Task completion time is a common metric in usability evaluation. To support the estimation of task completion time, model-based evaluation typically involves the modeling or reusing existing models of user operations' time cost. For example, Fitts' law [90] has long been used to model the time costs of users' object-pointing operations [180]. It has been extended to the time cost modeling in related mouse movement tasks, such as trajectory-based tasks [3] and object-pointing tasks with multi-scale navigation [108].

With models in model-based evaluation, redesign and optimization of user interfaces can be conducted computationally and swiftly, as demonstrated in the computational interaction literature [203]. Gajos and Weld propose the SUPPLE system for automatic user interface generation through constrained optimization [98]. They model the interface with functional specifications of the interface functionality. They optimize the navigation cost, given a model of the device showing the interface and the users' usage trace in the interface. Zhang and Zhai propose a card-playing model for information retrieval interfaces and demonstrate adaptive optimization of the interface according to the screen size and user's interaction sequence [295]. Todi et al. propose an adaptive user interface technique that uses Fitts' law to estimate user interaction cost and reinforcement learning to plan sequences of adaptions [259].

Like this line of research, Chapter 5 uses model-based evaluation to optimize interfaces, with estimated task completion time being the metric. Meanwhile, our model is dedicated to the scenario of quality assurance in interactive machine learning. While evaluating interactive machine learning systems is generally challenging [33], our work suggests that some of the most laborious routine processes in interactive machine learning, such as quality assurance, can be modeled effectively and evaluated efficiently.

A limitation of model-based evaluation methods is their restriction to routine and well-defined user tasks [133]. For exploratory tasks where the goal is not well-defined and requires users' complex reasoning procedures and creativity, it is intractable for the evaluators to model the usage procedures. This problem is absent in data extraction and data labeling tasks, both of which have well-defined goals and relatively predictable usage procedures. Thus, we can derive a model of the user operation sequences in Chapter 5.

"Without data, you're just another person with an opinion."

— William Edwards Deming

3 Data Extraction from Historical Visualizations with Machine Assistance

3.1 Introduction

Historical visualizations contain valuable historical datasets yet to be extracted from their scanned bitmap images. Extracting a dataset manually is laborious as it requires tedious measurements. Many attempts have been made to extract data from contemporary computer-generated visualizations (see Section 2.2.2). Some use hand-crafted image processing algorithms, while others use machine learning to derive models for recognizing visual objects that encode data.

However, unlike computer-generated imagery, a historical visualization image typically features a unique visual design. The hand-drawn nature of the visualization and the deterioration of and the damages to the paper media pose further challenges to recovering the data from historical visualization images. In addition, the extracted data is expected to be an accurate reusable benchmark. Inaccuracies in the extracted data will lead to the propagation of errors in downstream applications. Our objective is to support efficient and effective data extraction from historical visualizations (Section 1.2 **O1**).

If there were a fully automated solution for the end-user, the developer might have to develop an algorithm for each image individually by programming or machine

Part of the research in this chapter is published as "Yu Zhang, Bob Coecke, and Min Chen. 2021. *MI3: Machine-Initiated Intelligent Interaction for Interactive Classification and Data Reconstruction.* ACM Transactions on Interactive Intelligent Systems 11, 3-4, Article 18 (Aug. 2021), 34 pages." [297]. It was presented at ACM Conference on Intelligent User Interfaces, 2022. The overall idea was developed by Yu Zhang and Min Chen through discussions. Yu Zhang conducted the implementation, interviews, and writing. Min Chen and Bob Coecke guided the work and revised the writing.

learning. Likely the algorithm would be unsuitable for other historical visualization images. Therefore, the cost of manually recovering data would likely be lower than programming and, ironically, would be almost the same as the cost of preparing a training dataset for a machine learning process.

A more effective solution would be an interactive intelligent system equipped with a collection of elementary algorithms that asks users questions intelligently and adapts its algorithms automatically to work with a given historical visualization image. It is not an idealized system that could recover data automatically from many different historical visualization images but a computerized assistant that can initiate intelligent interactions with users to adapt itself to each specific task. The design goal of "intelligent interaction" is to reduce user operations and improve efficiency.

This chapter presents a generic approach for **machine-initiated intelligent interaction (MI3)**. The approach is governed by an iterative machine learning framework that features *algorithmic sampling (active sampling)* for dynamically acquiring labels from the user and *algorithmic default labeling (label propagation)* for maximizing the informational value of the user's inputs. Our approach can be seen as a mixed-initiative approach, according to Horvitz's definition of mixed-initiative systems as systems that "support an efficient, natural interleaving of contributions by users and automated services" [124].

Our approach consists of a collection of image processing algorithms to accommodate the needs for detecting different types of visual objects. Through the preprocessing by image processing algorithms, we convert data extraction problems to classification tasks, which are instances of data labeling tasks discussed in Chapter 4.

The goal of the MI3 approach is to perform data extraction tasks efficiently with as few user operations as possible. Using machine learning to train a model is primarily for supporting the task at hand rather than for deriving a model to be reused for many other visualization images since there are few similar images.

The main contributions of this chapter include:

- a **generic approach** to data extraction from historical visualizations that converts it to classification tasks and provides machine assistance,
- a demonstration of actualizing the approach through the design of reusable data extraction **workflows** and the implementation of data extraction **systems**, and
- a quantitative **simulation-based evaluation** of the approach in three workflows and a **domain expert interview** on the usage scenarios of data extraction.

3.2 Overview of the Machine-Aided Approach

This section first describes an example case that illustrates the challenges in a class of data extraction applications. We then outline a generic approach for data extraction.

3.2.1 Examples of Technical Challenges

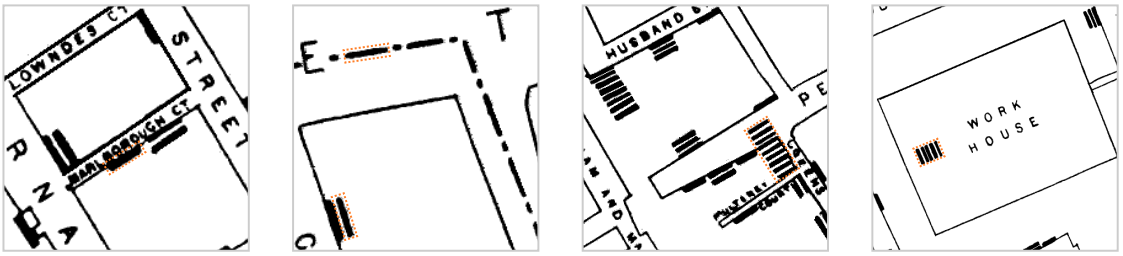


Figure 3.1: The cholera map [250] shows the number of fatalities in the 1854 London Cholera Outbreak with a discrete bar chart. The fatalities are distributed at more than 300 locations on the map and visualized as stacked blocks. The right figure illustrates the visual encoding of the cholera map. A bar aggregates the fatalities at a location, where the number of blocks denotes the number of fatalities. A bar typically sits on a base with a root point denoting its location information.

The 19th century witnessed the boom of visualization techniques. Among many well-known historical visualizations created at that time, John Snow's cholera map created in 1854 [250] provides a telling example to illustrate the challenges to extract data from historical visualization images. As shown in Fig. 3.1, the thematic map depicts the spatial distribution of fatalities in an area during the 1854 London cholera outbreak. Because of the prevalence of historical thematic maps, the following uses Snow's famous cholera map as an example.

The cholera map encodes a precious historical dataset, which can be stored as an array of records, each of which is a tuple $(x, y, \#victims)$. The goal of data extraction is thus to scan the map, identify the location (x, y) of each bar with at least one block, and count the number of blocks as the value of $\#victims$. As illustrated on the right of Fig. 3.1, the identification of each location (x, y) requires several object recognition processes to identify the corresponding bar, the blocks that form the bar, the base that determines the first block, and the root position (i.e., a consistently-defined position in the first block).

The cholera map consists of 579 blocks that form 321 bars. Manually acquiring the values for the 321 records will be tedious and time-consuming. Even with an optimistic estimation, if one were to spend 10 seconds navigating to a bar, 10 seconds counting the number of blocks in the bar, 20 seconds measuring the location, and 10 seconds typing in the values into a spreadsheet, it would take some 50 seconds per bar and more than 4 hours to extract the dataset.



(a) Segmentation difficulty (b) Similar non-block object (c) Ambiguous grouping (d) No baseline

Figure 3.2: Some examples of challenging cases in processing the cholera map image [250]: (a) A block conglutinates with the background; (b) A segment of the dashed line may be confused for a block; (c) The grouping of blocks is ambiguous; (d) Blocks are not distributed close to a baseline.

Naturally, one would like to automate this process as much as possible. However, data extraction from historical visualizations poses several challenges. Many such visualizations were hand-drawn, often featuring unique visual designs and suffering from poor image quality. Any general-purpose algorithms for detecting blocks, bars, bases, or roots may deliver erroneous results when encountering difficult patterns such as those shown in Fig. 3.2. In Fig. 3.2(a), a block conglutinates with the baseline and the letters on the other side. In Fig. 3.2(b), a segment in a dashed line can easily be confused with a block. In Fig. 3.2(c), the grouping of blocks into bars is ambiguous, even for humans. At a glance, the highlighted pattern may be interpreted either as a bar with nine blocks or as two bars with four and five blocks, respectively. It requires careful observation of other bars in the neighborhood and some logical reasoning in order for one to conclude that this is a bar of nine blocks. In Fig. 3.2(d), a bar does not have a baseline because it is not associated with a street location. Hence, the identity of its first block is ambiguous, and so is its root position.

While one may anticipate the potential of using machine learning to train a sequence of models for detecting blocks, grouping blocks into bars, and finding the root of each bar respectively, the unavailability of a sufficient number of training data objects poses another critical challenge. Many historical visualizations, including the cholera map, feature unconventional visual designs. Given such a visualization image, the data objects (e.g., blocks, bars, or roots) that may be prelabeled to aid a supervised learning process would constitute a very sparsely sampled and possibly biased training dataset if there were other visualizations drawn with the same visual design. Therefore, prelabeling the 579 blocks, 321 bars, and 321 roots would unlikely cost less than manually measuring the 321 tuples $(x, y, \#victims)$ and typing them into a spreadsheet.

Hence, it is highly desirable to find an approach that is neither totally automated nor totally manual. Ideally, a software system enabling such an approach encodes basic knowledge about a family of charts (e.g., bar charts, discrete bar charts, pictogram

bar charts). It asks a user questions about the difficulties and variations specific to a given visualization (e.g., noise and distortion, and root definition) and dynamically learns from the user's answers to improve its ability to handle similar difficulties and variations. Like an intelligent assistant, it initiates interactions intelligently and learns dynamically. This desire motivates the development of the MI3 approach, where MI3 stands for "Machine-Initiated Intelligent Interaction".

3.2.2 A Generic Workflow for Interactive Classification

The goal of an MI3 workflow is to perform an interactive classification task, which is an instance of data labeling tasks in Chapter 4. Many detection problems in data extraction are inherently classification problems. For example, given a collection of pixel-based objects in the cholera map, differentiating those blocks that represent fatalities from other shapes is a typical classification problem. Even when a detection problem may not immediately appear as a classification problem, it can usually be decomposed into a classification problem plus some preprocessing and postprocessing. For example, the problem of grouping blocks into a bar can be transformed into a classification problem for determining whether two blocks are related (i.e., belong to the same bar), together with a preprocessing step for compiling candidates of block pairs and a postprocessing step for grouping those related blocks. Both the preprocessing and postprocessing steps can be reliably computed using predefined algorithms without user intervention. By focusing on classification problems, the MI3 approach applies to different detection problems in data extraction pipelines.

Fig. 3.3 depicts the MI3 approach as a generic workflow. Let $\mathbf{D}_0$ be a set of n unlabeled data objectives at step 0 (i.e., before the iteration starts). The goal of the workflow is to deliver a labeled dataset $\binom{\mathbf{D}_0}{\mathbf{L}_0}$ after $k+1$ iteration steps such that the amount of human effort in these $k+1$ steps is less than that for labeling all n data objects manually. This chapter uses the number of user operations to approximate the amount of human effort.

In the workflow, *data preprocessing* is an optional process, and it is used when the raw data has not yet provided a set of data objects to be labeled. For example, given the cholera map image, one may use a preprocessing step to extract all color connected pixel components (i.e., all connected black pixel groups and all connected white pixel groups) as candidates for blocks. Typically, this preprocessing step is purely automatic. The subsequent processes will classify these candidates into correct categories, such as "true blocks" and "false blocks". It is thus essential in practice for this preprocessing step to minimize false negatives.

As all data objects in $\mathbf{D}_0$ are unlabeled, the core of the workflow is an iterative loop based on active learning (according to its broad definition in Section 2.3.3). Consider the $k+1$-th iteration as an instance. The data flows through various component

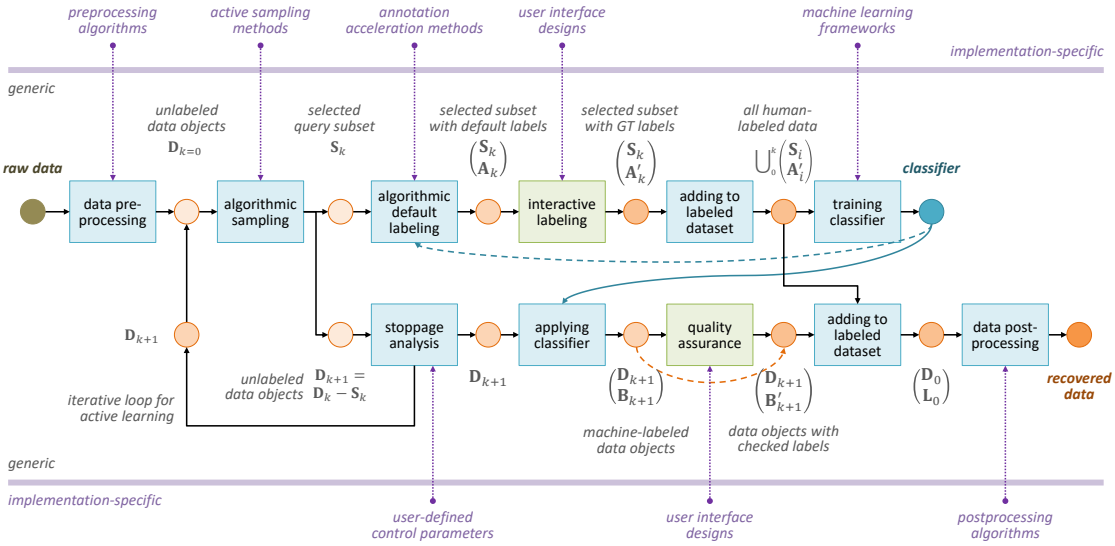


Figure 3.3: The generic MI3 workflow: The blue blocks denote algorithmic steps, and the green blocks denote interactive steps. The optional data preprocessing step transforms the raw data from the original problem into a classification task for a set of unlabeled data objects $\mathbf{D}_0$. We consider the $k+1$-th iteration of the active learning as an instance. Here, by "active learning", we refer to its general definition that the system actively seeks decisions from the user to aid the ML process. The algorithmic sampling step selects the data objects $\mathbf{S}_k$ to be labeled. The algorithmic default labeling step assigns default label annotation $\mathbf{A}_k$ for data objects $\mathbf{S}_k$. In the interactive labeling step, the user corrects the mislabeled data objects in the interface and produces ground truth (GT) labels $\mathbf{A}'_k$. The confirmed label annotations $\mathbf{A}'_k$ are then added to the labeled dataset. An interim classifier is trained with the partially labeled dataset, which can be used to compute default labels in the next iteration. The stoppage analysis step checks whether the interactive classification stage should stop. If not, the next session of interactive classification starts. Else, the interim classifier can be applied to label the remaining unlabeled data objects $\mathbf{D}_{k+1}$ with labels $\mathbf{B}_{k+1}$. The user goes through a quality assurance step to verify the labels for these data objects. The data objects $\mathbf{D}_{k+1}$ and verified labels $\mathbf{B}'_{k+1}$ are then added to the labeled dataset and form the labels $\mathbf{L}_0$ for the whole data object set $\mathbf{D}_0$. An optional postprocessing step transforms the labeled data objects into the desired final output data structure. The MI3 workflow is generic, and its components can be implemented with application-specific algorithms.

processes as follows:

- The process *algorithmic sampling* selects a subset of data objects $\mathbf{S}_k \subseteq \mathbf{D}_k$. While a simple solution can be random sampling, a more sophisticated solution can select data objects that can effectively inform the classifier learning.

- The process *algorithmic default labeling* assigns a tentative label to each object in $\mathbf{S}_k$. We denote this interim labeled subset as $\binom{\mathbf{S}_k}{\mathbf{A}_k}$. It is helpful for those tentative labels to be as correct as possible. All incorrect labels will have to be manually corrected by the succeeding process, hence incurring more user operations.

- The process *interactive labeling* then initiates an interaction session, asking the user to check the interim labeled subset and make a correction if necessary. This results in a correctly-labeled subset $\binom{\mathbf{S}_k}{\mathbf{A}'_k}$. The main criterion for designing an effective user interface is to enable the checking of the tentative labels and correcting any mistakes with minimal human effort. This chapter uses the number of user operations to approximate human effort.

- In the process *adding to labeled dataset*, the labeled subset $\binom{\mathbf{S}_k}{\mathbf{A}'_k}$ is then combined with all labeled subsets in the previous k iterations, resulting in a larger set of labeled data objects $\bigcup_0^k \binom{\mathbf{S}_i}{\mathbf{A}'_i}$, which can be used to train a classifier while forming part of the classification results that the workflow is designed to deliver. Here we define the union of a series of set pairs as the pair of two sets, each resulting from the union of the corresponding series of sets.

- The process *training classifier* is an automated ML process that uses the labeled data objects $\bigcup_0^k \binom{\mathbf{S}_i}{\mathbf{A}'_i}$ to train a classification model. Because this training process is iteratively invoked with bigger and bigger training datasets, the ML models are expected to become better and better. The interim model can be used to classify unlabeled data objects in the processes of *algorithmic default labeling* and *applying classifier*.

- The process *stoppage analysis* makes an algorithmic decision as to whether the trained model obtained after the $k+1$ iteration steps is good enough for classifying unlabeled dataset $\mathbf{D}_{k+1}$. This decision can be made using the information collected in the previous $k+1$ iterations, e.g., the testing measures in the process *training classifier*, the percentage of the labels that need to be corrected in the process *interactive labeling*, and some statistical measures about the labeled data and unlabeled data. If the trained classifier is equipped with uncertainty analysis, there can be an additional *stoppage analysis* process after the process of *applying classifier*.

- The process *applying classifier* is invoked once the *stoppage analysis* gives the green light. As $\mathbf{S}_k$ is selected from $\mathbf{D}_k$, the set of remaining data objects yet to be labeled is denoted as $\mathbf{D}_{k+1}$. Using the trained model, the process labels all the data objects in $\mathbf{D}_{k+1}$, resulting in a labeled dataset $\binom{\mathbf{D}_{k+1}}{\mathbf{B}_{k+1}}$.

- The process of *quality assurance* is necessary for any application that demands a very high quality of data extraction. It is an interactive process for a user to inspect the machine-labeled data objects in $\binom{\mathbf{D}_{k+1}}{\mathbf{B}_{k+1}}$ and correct all errors found. This results in a checked and corrected dataset $\binom{\mathbf{D}_{k+1}}{\mathbf{B}'_{k+1}}$.

- The second process of *adding to labeled dataset* combines $\binom{\mathbf{D}_{k+1}}{\mathbf{B}'_{k+1}}$ and $\bigcup_0^k \binom{\mathbf{S}_i}{\mathbf{A}'_i}$ to form the final set of labeled data objects, which is denoted as $\binom{\mathbf{D}_0}{\mathbf{L}_0}$.

Similar to *data preprocessing*, *data postprocessing* is an optional process for transforming the labeled data objects $\binom{\mathbf{D}_0}{\mathbf{L}_0}$ to those in an intended data structure and format. In some applications, an MI3 workflow may be followed by another MI3 workflow. We will see examples of *data preprocessing*, *data postprocessing*, and multiple pipelines in the following sections.

3.3 Instantiating Machine-Aided Workflows in Practice

This section uses the process of designing a software system for extracting data from spatial visualizations to showcase how to design an interactive classification process following the generic MI3 workflow in practice. The cholera map shown in Fig. 3.1 typifies such a visual representation. The data extraction system consists of three workflows, all of which were designed and implemented following the generic MI3 workflow. They are workflows for (1) detecting objects (e.g., detecting blocks in the cholera map), (2) grouping components (e.g., detecting bars in the cholera map), and (3) determining key positions (e.g., detecting roots in the cholera map).

3.3.1 Transforming a Detection Problem to a Classification Problem

To benefit from the generic MI3 approach, one needs to transform various detection problems, such as detecting objects, grouping components, and determining key positions, into classification problems. As mentioned in Fig. 3.3, we can use a preprocessing step to generate a list of candidates representing potentially correct decisions. These candidates then become the inputs of a classification workflow, which categorizes these candidates into different label classes.

For example, when the first MI3 workflow is activated for detecting blocks in the cholera map, the preprocessing step may construct a list of candidates, each of which is a group of connected pixels. Thus, the function of the workflow is to label each candidate as a valid "block" representing a victim or a "non-block". All candidates labeled as "block" are then passed to the next workflow for grouping these blocks into bars.

When the second MI3 workflow is activated for grouping the detected blocks into bars, the preprocessing step may generate a list of pairwise relations, each indicating that a pair of blocks may potentially belong to the same bar. These candidate relations are then fed into the interactive classification workflow for labeling each candidate as "yes" for belonging to the same bar and "no" otherwise. In this case, a postprocessing step is necessary for transforming the results of the classification problem into the

results of the original grouping problem. This can be achieved with a simple algorithm converting detected blocks and confirmed relations to a list of bars, each consisting of a group of blocks linked with the confirmed relations. This postprocessing step can also count the blocks in each bar, resulting in the data value $\#victims$ for each bar.

When the third MI3 workflow is activated for determining the root positions of the bars, the preprocessing step may generate a set of potential positions for each bar, such as the center of the bar and that of each block in the bar, the middle point of each edge of the bar and its blocks, and the corners of the bar and its blocks. The classification workflow then labels the candidate positions as "root" and "non-root" while ensuring exactly one root per bar. A postprocessing step then converts the local root position relative to the bar to a global position (x, y) relative to the original visualization image. In combination with the data value $\#victims$ for each bar, the system generates a list of tuples in the form of $(x, y, \#victims)$.

If the user requires the actual geographical location (e.g., longitude and latitude or other geodetic systems) for each tuple, this can be done trivially using an image-specific coordinate transformation as a postprocessing step.

3.3.2 Variations of Workflows

The design and implementation of an MI3 workflow in practice depend on many factors, such as the technical availability of various algorithmic components, the knowledge and skills of the developers, the cost and time constraints of the software life cycle, and many other application-specific requirements. In general, one can consider the development as an agile software engineering approach, with a gradual introduction of better algorithms, techniques, or user interfaces for different processes in Fig. 3.3. One may consider some typical variations of the processes in the interactive classification part of MI3 workflows:

- Variations of *algorithmic sampling* — Perhaps the most naive sampling method, which is denoted as **NS**, is to fetch data objects to be labeled simply according to their order in the input dataset $\mathbf{D}_k$. More commonly used in ML, *random sampling* (**RS**) is considered as the baseline sampling method, which selects data objects for labeling in a stochastic order. The approach of *algorithmic sampling* (**AS**) selects data objects according to a predefined metric that predicts the potential benefit of labeling a data object to the learning process. Such a metric may assess the potential benefit according to the uncertainty of the label [38], the diversity of the samples [281], clustering information [196, 281], and the expected performance of the classifier [109]. We have used two methods, Lewis and Catlett's uncertainty-based sampling [167] and Xu et al.'s sampling method according to weighted uncertainty, cluster density, and sample diversity

score [281]. Section 4.4.3, introduces more instantiations of algorithmic sampling, which includes not only active learning methods but also alternatives such as using clustering algorithms for sampling.

- Variations of *algorithmic default labeling* — When data objects are presented in a user interface for dynamic labeling, the system may assign a default label to each data object. If the system can predict some labels correctly, this can reduce the number of user operations in labeling the data objects. A most naive approach is *no default* labeling (**ND**). An alternative approach is *random default* labeling (**RD**), which assigns a data object to one of the label classes in a stochastic manner. The approach of *algorithmic default* labeling (**AD**) uses a model to predict the label of a data object. For example, one may use the interim model during an ML process to label a data object. Alternatively, one may use a label propagation algorithm (e.g., self-training [286], co-training [30], graph-based label propagation [300, 305]) to predict the label of a data object according to those data objects that have already been labeled. In this chapter, we have instantiated our MI3 workflows with the options of using the interim model (**AD (Interim)**) as well as a graph-based label propagation algorithm (**AD (Graph)**) [300] for default labeling. In Section 4.4.3, we introduce more instantiations of default labeling.

- Variations of *interactive labeling* — The approach of active learning represents a small portion of all research papers published on ML. One can naturally consider that the baseline approach in ML is to prelabel all data objects in the training and testing datasets without any interactive labeling during an ML process. This baseline approach is denoted as *prelabeling* (**PL**). Since the MI3 approach is designed for active learning, the process of interactive labeling (**IL**) is expected to be present in almost all MI3 workflows. Different designs of the user interface for interactive labeling will impact an MI3 workflow differently. It is desirable to explore many design options. We mainly focused on using the simplest interaction modality of button clicking to minimize the time and cognitive load per interaction.

- Variations of *training classifier* — We consider that the baseline is without any automatic and semi-automatic ML, and all data objects will be labeled manually. We denote this *brute force* approach as **BF** while denoting the fully automatic and semi-automatic ML approaches as **ML**. In practice, there are many design options for an ML pipeline. The MI3 approach does not impose any restrictions. For example, MI3 workflows can be instantiated with different ML frameworks for classification, e.g., variations of neural networks, decision trees, random forests, support vector machines, and Bayesian networks. In this chapter, we have used

decision tree and the transductive model learned in label propagation [300] as two options for the process *training classifier*. In Section 4.4.3, we introduce more instantiations of model training.

In Section 3.4, we will provide further technical details of the algorithms implemented in a software prototype built based on the MI3 approach. In Section 3.5, we will compare how different combinations of the above variations may impact the performance of the implemented MI3 workflows.

3.4 Data Extraction from Historical Visualization Images

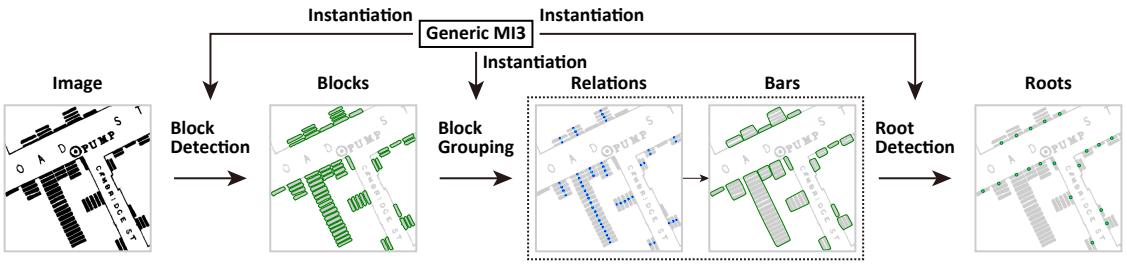


Figure 3.4: A data extraction procedure for spatial data visualization with three subroutines: block detection, block grouping, and root detection. Each of the three subtasks is an interactive classification task and is solved with an instantiation of the generic MI3 workflow. **(1) Block Detection**: Given an input image, we use connected components detection as a preprocessing to convert block detection into a classification task for connected components. **(2) Block Grouping**: With the detected blocks, we build a graph of block relations and classify the existence of relations to determine which pairs of blocks should be grouped in the same bar. The confirmed relations are used to compile the detection of bars. **(3) Root Detection**: For each bar, we generate candidate root points. Therefore, the task of determining the key point is transformed into the classification of candidate roots.

To validate the feasibility of the MI3 approach and to study and evaluate different design options in realizing MI3 workflows, we have developed a prototype software system for extracting data from a family of visual representations, which are composed of a collection of spatially distributed similar data objects. Each data object depicts a data tuple $(x, y, \#value)$. The spatial location (x, y) may be geographically meaningful and related to a background map or image. The variable $\#value$ may be encoded as the size of the data object, the number of components inside the data object, or other attributes. The cholera map shown in Fig. 3.1 exemplifies such a visual representation. Fig. 3.4 shows the procedure of data extraction from the discrete bar chart. We decompose the data extraction task into three subtasks: block detection, block grouping, and root detection. In the following subsections, we first detail the preprocessing steps that transform the three subtasks into classification problems. We then describe the active learning components shared by the MI3 workflows for the

three subtasks. We introduce an implementation of MI3 workflows in an interactive intelligent system. Later, Section 4.5 demonstrates that the interactive intelligent system can be reproduced with OneLabeler.

3.4.1 Block Detection Preprocessing

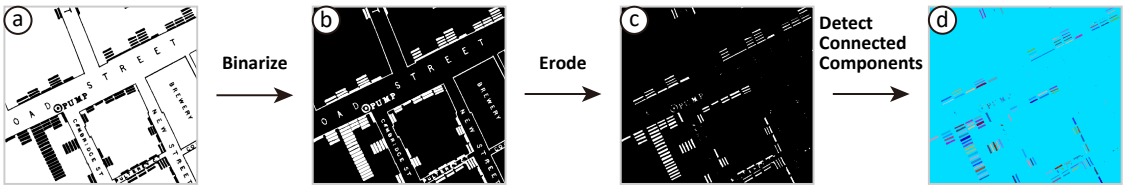


Figure 3.5: Candidate block detection process on part of the cholera map [250]: (a) Input part of the cholera map; (b) Binarize the image to reduce color variations; (c) Erode the image to reduce conglutination; (d) Detect connected components.

Following the generic MI3 workflow, we need to transform the block detection problem into a classification problem. For this transformation, we need first to generate candidate data objects (i.e., in this case, connected components) to be classified and then compute the feature representation of each data object to facilitate the training of the interim classifier.

Candidate Data Object Detection: We design a preprocessing algorithm, as illustrated in Fig. 3.5. With the observation that blocks differ from the background in color, we let connected components serve as candidate blocks. The candidate block detection procedure is as follows:

1. Binarize the image with Otsu thresholding to eliminate minor color variations (Fig. 3.5(b)).
2. Erode the binary image with 3×3 cross kernel for two iterations to reduce the conglutination of elements (Fig. 3.5(c)).
3. Detect connected components (Fig. 3.5(d)).
4. Dilate each connected component separately using the cross kernel for two iterations to compensate for the size shrinkage during erosion.

Feature Computation: For each candidate block, we compute two sets of features: 13 appearance features and 14 neighborhood features, which sum to 27 features. Appearance features capture the visual appearance of the candidate block. Neighborhood features capture the image features of the candidate block's neighborhood.

We compute size features, shape features, and color features as appearance features to capture the appearance of a candidate block. Size features include *width*, *height*, *axis-aligned width*, *axis-aligned height*, and *area*, which are all measured in

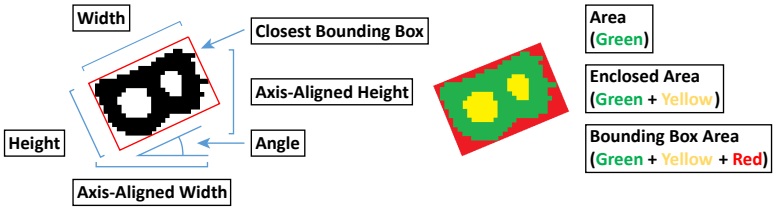


Figure 3.6: Measurements related to appearance features of candidate blocks. The left shows the size features and the angle. The right shows area definitions involved in the computation of shape features.

pixels. Shape features include $solidity = \frac{area}{enclosed\ area}$, $convexity = \frac{area}{convex\ hull\ area}$, $extent = \frac{area}{bounding\ box\ area}$, $aspect\ ratio = \frac{width}{height}$, and *angle*. Fig. 3.6 shows visual attributes used as size features and attributes used for computing shape features. Color features include *r*, *g*, and *b* computed by averaging the color of all the pixels. In total, 13 appearance features are computed.

The 14 neighborhood features are mainly based on the distribution of pixel colors in the candidate block's neighborhood. Multiple neighborhood definitions are considered in the computation. Below describes the details of neighborhood features.

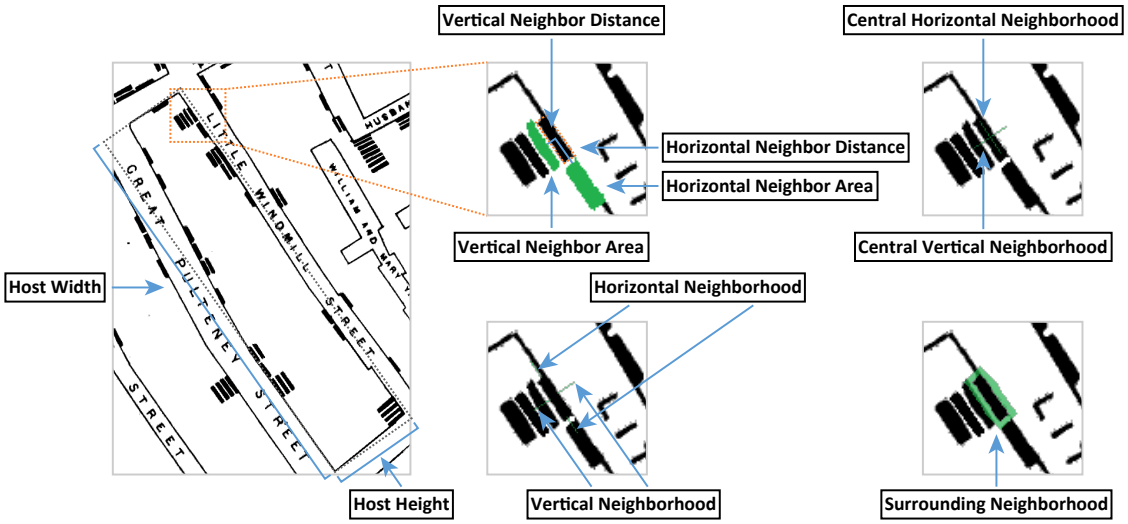


Figure 3.7: Neighborhood features and measurements. Horizontal/Vertical neighborhood refers to the union of 10 pixels horizontally/vertically extending outward the candidate block along its horizontal/vertical axis. The measurements of central horizontal and vertical neighborhoods are similar. Surrounding neighborhood refers to the union of pixels within 5 pixels' distance from the border of the candidate block.

In some cases, true detections and false detections of blocks are not differentiable by appearance. For example, dashes in dashed lines and blocks look identical on the cholera map. To handle such cases, we compute neighborhood features to capture the image features in the candidate blocks' neighborhoods. Fig. 3.7 illustrates some of the neighborhood features and neighborhood measurements used for feature

computation. For conciseness, we refer to the two symmetry axes of a candidate block's bounding box as the candidate block's horizontal and vertical axis in the following. The horizontal/vertical axis is the one extending along the long/short edge of the bounding box.

Firstly, we compute neighbor distribution features to depict the distribution of other connected components around the candidate block. The *horizontal/vertical neighbor distance* captures the distance to the nearest component (with the background excluded) in the horizontal/vertical direction. The *surrounding neighbor number* feature denotes the number of adjacent components to the candidate block.

Secondly, We compute neighbor features to capture the properties of neighboring connected components of the candidate block. *Horizontal/Vertical neighbor area* measures the area of the closest neighboring connected component (with the background excluded) in the horizontal/vertical direction. *Host width*, *host height*, *host area*, and *host solidity* measure the width, height, area, and solidity of the connected components containing the candidate block.

Thirdly, we compute neighborhood foreground rate features to capture the distribution of foreground pixels (i.e., white pixels in the binarized image) in the candidate block's neighborhood. *Surrounding foreground rate* is computed as the rate of foreground pixels within 5 pixels' distance from the candidate block's border. *Horizontal/Vertical foreground rate* is computed as the rate of foreground pixels within 10 pixels' distance from the candidate block's border along the horizontal/vertical axis. Similarly, *central horizontal/vertical foreground rate* measures the rate of foreground pixels along the two axes but concerns the pixels within 10 pixels' distance from the candidate block's center instead of the border.

3.4.2 Block Grouping Preprocessing

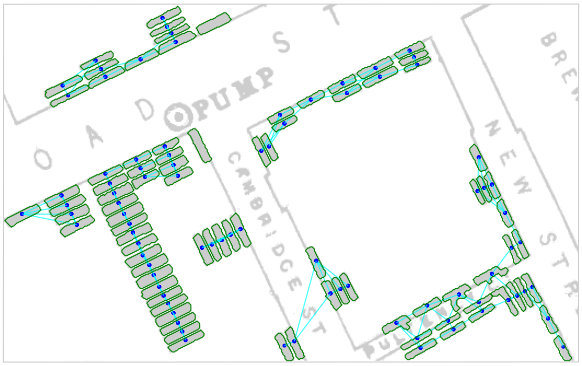


Figure 3.8: Candidate neighboring relation detection result on part of the cholera map [250]. The blocks are denoted with green contours. The blue dots show the center of each block. The detected candidate neighboring relations are shown as cyan lines linking the centers of the two blocks involved.

To aggregate the number of victims according to location, we need to group neighboring blocks into bars. We reduce the grouping problem to determining which pairs of blocks are neighbors. If we regard each block as a node and each possible neighboring relation as an edge, the grouping problem is transformed into graph edge classification, which fits the MI3 workflow. Fig. 3.8 highlights the detected blocks and candidate neighboring relations.

Candidate Data Object Detection: Each pair of blocks is potentially a pair of neighboring blocks. Given n blocks, there are in total $\frac{1}{2}(n^2 - n)$ candidate relations. In practice, the quadratically exploding number of data objects poses a computation challenge for algorithms. Therefore, we design two heuristic filtering rules as follows:

1. There is no relation between a pair of blocks when they are not mutually the top four closest to the other block among all the blocks.

2. There is no relation between a pair of blocks when the distance between the two blocks is larger than the length of the longest axis of all blocks.

Feature Computation: For each candidate neighboring relation, we compute *position distance*, *color distance*, and *overall distance* as the features. For a pair of blocks, the *position distance* is defined as the Euclidean distance between the position of the two blocks (x_1, y_1), (x_2, y_2) in pixels. For a pair of blocks, the *color distance* is defined as the Euclidean distance between the two blocks in the (r, g, b) space. The *overall distance* is defined as the root mean square of the *position distance* and *color distance*.

3.4.3 Root Detection Preprocessing

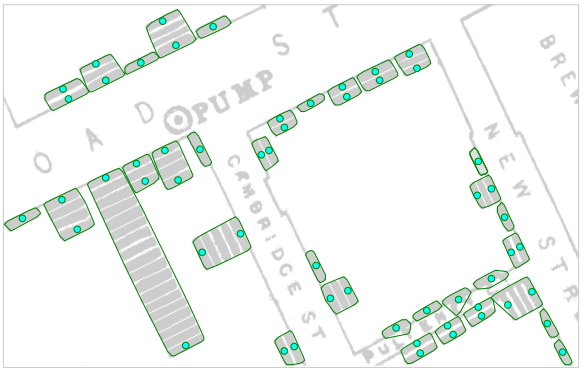


Figure 3.9: Candidate root detection result on part of the cholera map [250]. The bars are denoted with green contours. Each candidate root is denoted with a cyan dot.

To extract the position information of a data object in the spatial visualization, we need to detect the data object's representative point. To align this problem to the MI3 workflow, we need to generate a set of candidate roots to be classified, where a "root" refers to the key point in the data object that corresponds to its position.

Candidate Data Object Detection: We simplify the root detection problem as finding the center of a block in a bar that represents the position. The root of a bar is an extreme point of the bar that is typically closest to the boundary lines in the image. Therefore, we take the center of the topmost and the bottommost blocks of the bar to be candidate roots. Fig. 3.9 shows the detected candidate roots on part of the cholera map.

Feature Computation: For each candidate root, we compute in total 9 neighborhood features. We observe that roots are typically close to boundary lines in the image and that roots of multiple bars may be aligned. Therefore, we reuse some of the neighborhood features of the block as root features, including *horizontal neighbor distance*, *surrounding neighbor number*, *surrounding foreground rate*, *horizontal foreground rate*, *host area* and *host solidity*. Besides, we compute a boolean feature *single* to denote whether the block containing the root is the only block in a bar. When a root is *single*, it is always a true detection. We also compute two additional neighbor distribution features: *distance to alignment line* and *distance to skeleton*. The two features are described below.

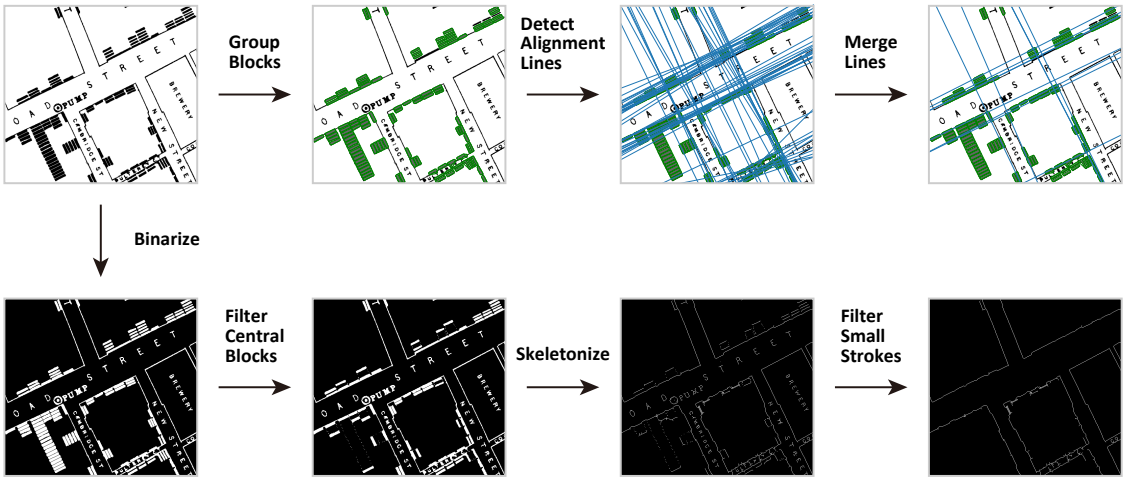


Figure 3.10: The first row shows the process of detecting alignment lines, which is used for computing the distances from candidate roots to alignment lines. The second row shows the process of detecting the skeletons, which is used for computing the distances from candidate roots to skeletons.

Distance to alignment line captures how well the candidate root aligns with other candidate roots. For each candidate root, we denote the horizontal axis of the block containing the root as the *alignment line* that it generates. For all the alignment lines generated by all the candidate roots, we merge the alignment lines that are close to each other. Quantitatively, we denote the distance between two lines to be the maximal distance from a point on a line within the image range to the other line. Two lines are merged when their distance is smaller than the minimum width and height of blocks. For the merged lines, we refit a line with linear regression of the positions of the roots that generate the lines. Then, we filter the lines that cannot be merged

with any other lines. This process of computing the alignment lines is shown in the first row of Fig. 3.10. With the merged lines (Fig. 3.10 top right), we can measure the *distance to alignment line* of each candidate root.

Distance to skeleton captures whether the candidate root is close to a boundary line in the image. As shown in the second row of Fig. 3.10, to compute this feature, we detect the skeleton of the binarized image using Guo and Hall's thinning algorithm [110]. To remove noise for skeleton detection, we further adopt two strategies. Firstly, we remove all the pixels occupied by blocks that do not contain candidate roots before the skeleton detection. In this way, these pixels will not be mistaken for the skeleton. Secondly, after the skeleton detection, we filter the detected skeleton strokes that are too small. We compute the connected components formed by the skeleton pixels using 8-connectivity and remove the connected components whose diagonal of the axis-aligned bounding box is shorter than twice the maximal width and height of blocks. Using the filtered skeleton, we measure the *distance to skeleton* of each candidate root.

3.4.4 Interactive Classification and Active Learning

In the previous subsections, we illustrate how to transform subtasks of data extraction into interactive classification problems that MI3 concerns. In the following, we introduce how the two major algorithmic components of MI3, algorithmic sampling and algorithmic default labeling, are instantiated in our implementation. We also introduce our prototype implementation of an MI3-based data extraction system, which shows how the major interactive component, interactive labeling, is instantiated. The implementations of algorithmic sampling, algorithmic default labeling, and interactive labeling are reused for all three subtasks.

Algorithmic Sampling (AS)

The *algorithmic sampling* process selects a set of data objects $S_k = (x_{k_1}, ..., x_{k_l})$ to be labeled interactively, where l is the number of data objects shown in the interface at a time. The selected data objects are to be given default labels (see Section 3.4.4), and the data objects and their default labels are to be presented to the user for label correction (see Section 3.4.4). The input to *algorithmic sampling* is unlabeled data objects $D_k = (x_1, x_2, ..., x_n)$ as shown in Fig. 3.3. We instantiate *algorithmic sampling* with two optional sampling methods: *entropy-based sampling* and *density, diversity, and entropy-based sampling* (abbreviated as **AS (ES)** and **AS (DDES)** respectively) based on two sampling methods in the literature [167, 281].

With *entropy-based sampling*, the metric of sample priority is defined as the uncertainty of the label [167]. The higher the label's uncertainty, the more the label information can inform the classifier learning process, thus resulting in less interaction cost. The uncertainty of an unlabeled data object x_i's label y_i is quantified by entropy

as $H(y_i|x_i) = -\sum_j p(y_i = j|x_i) log(p(y_i = j|x_i))$. When l instances are to be presented to the user, the top l unlabeled instances that maximize the entropy are sampled. In reality, we do not have access to the true posterior probability $p(y|x)$ used in entropy calculation, and therefore it has to be approximated. When the interim model trained in the *training classifier* stage is a probability-based classifier, we directly use the posterior probability estimated by the classifier as the approximate true posterior probability. When the interim model is not a probability-based classifier, e.g., a decision tree, we use Zhou et al.'s algorithm [300] to train an accompanying probability-based classifier that serves as a posterior probability estimator.

Entropy-based sampling only aims to maximize the uncertainty of the samples, while *density, diversity, and entropy-based sampling* sets two additional goals for optimization [281]. It requires that in each iteration, the sampled instances should be distant from each other (diverse), and the sampled instances should be representative in the feature space (dense). To this aim, the scoring function of each data object's priority is defined as $score(x_i) = (1-\alpha-\beta)H(y_i|x_i) + \alpha\sum_{j=1}^{n}\frac{1}{1+d(x_i,x_j)} + \beta\sum_{j\in S_k}(1 - \frac{1}{1+d(x_i,x_j)})$. It maximizes a weighted sum of label entropy, average inverse distance to other unlabeled data objects, and distance to the data objects already sampled in this batch. The data object with the highest score is added to the sampled set S_k, and sampling is conducted iteratively until $|S_k| = l$. In our implementation, we set $\alpha = \beta = \frac{1}{3}$, and the distance function d to measure Euclidean distance.

Algorithmic Default labeling (AD)

Through the *algorithmic default labeling* process, default labels are assigned to sampled data objects, which saves the user's effort for labeling. We instantiate *algorithmic default labeling* with two implementations, including *interim-model-based default* (abbreviated as **AD (Interim)**) and *graph-based default* (abbreviated as **AD (Graph)**).

With *interim-model-based default*, an interim classification model is trained and updated each time a batch of newly labeled data objects are added to the labeled set. The interim model predicts the default labels for the instances to be presented to the user in the current iteration.

The idea of *graph-based default* is that data objects close to each other in the feature space tend to have the same label, and therefore labeled data objects can propagate their labels to unlabeled data objects. The closer a pair of instances are, the more likely the label can propagate between them. We adapt Zhou et al.'s label propagation algorithm [300] for our implementation of the *graph-based default* method, which is initialized with Algorithm 1 and uses Algorithm 2 to update incrementally in the iterative labeling sessions.

Quantitatively, matrix S in Algorithm 1 denotes the probability of propagation. $S_{i,j}$ is the probability that the label of data object i propagates to data object j in one

Algorithm 1 Transition Matrix Computation

Require: instances $X = \{x_1, ..., x_n\}$
Ensure: transition matrix $K_{n\times n}$
1: $W \leftarrow [I(i \neq j)exp(-\frac{||x_i - x_j||^2}{2\sigma^2})]_{n\times n}$
2: $D \leftarrow [I(i = j)\sum_{k=1}^{n} W_{i,k}]_{n\times n}$
3: $S \leftarrow D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$
4: $K \leftarrow (I - \alpha S)^{-1}$
5: **return** K

propagation iteration. This probability is based on the normalized distance between data objects i and j defined as $S_{i,j} = \frac{W_{i,j}}{D_{i,i}}$. $W_{i,j} = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$ is the unnormalized distance between data objects i and j (where $i \neq j$). $D_{i,i} = \sum_{k=1}^{n} W_{i,k}$ is the sum of unnormalized distance between data object i and all the other data objects. n is the total number of data objects, no matter labeled or not. The propagation process is run for multiple iterations until convergence. Transition matrix K denotes the transformation that transforms the initial label distribution to the label distribution at the convergent state.

Algorithm 2 Incremental Label Propagation

Require: transition matrix $K_{n\times n}$, labels of newly labeled instances $y_{q_1}, ..., y_{q_l}$, label distribution in the last iteration $F^t_{n\times c}$ (initialized as $F^0 = 0_{n\times c}$ in the first iteration)
Ensure: updated label distribution $F^{t+1}_{n\times c}$
1: **for** $q \in \{q_1, ..., q_l\}$ **do**
2: $\Delta F \leftarrow [I(j = y_q)K_{i,q}]_{n\times c}$
3: $F^{t+1} \leftarrow F^t + \Delta F$
4: **end for**
5: **return** F^{t+1}

Each time the user confirms or corrects a batch of l data objects, the algorithm propagates the labels using the transition matrix K, as shown in Algorithm 2 to update the default labels incrementally. In the label distribution matrix $F^t_{n\times c}$, element $F^t_{i,j}$ denotes the estimated probability of whether data object i belongs to class j at the start of the t-th iteration of interactive labeling. c is the total number of classes in the classification. Using the updated label distribution F^{t+1}, the algorithm computes the default labels by maximal likelihood.

The Interactive Labeling Interface

In the interface, as shown in Fig. 3.11, the user first uploads a visualization image and specifies the type of chart that determines the preprocessing method in the Control Toolbar (Fig. 3.11(a)). Then, the user clicks the start button on the header of the Image View (Fig. 3.11(b)) to start the extraction. If needed, the user can switch between different implementations of the MI3 algorithmic components with a popup

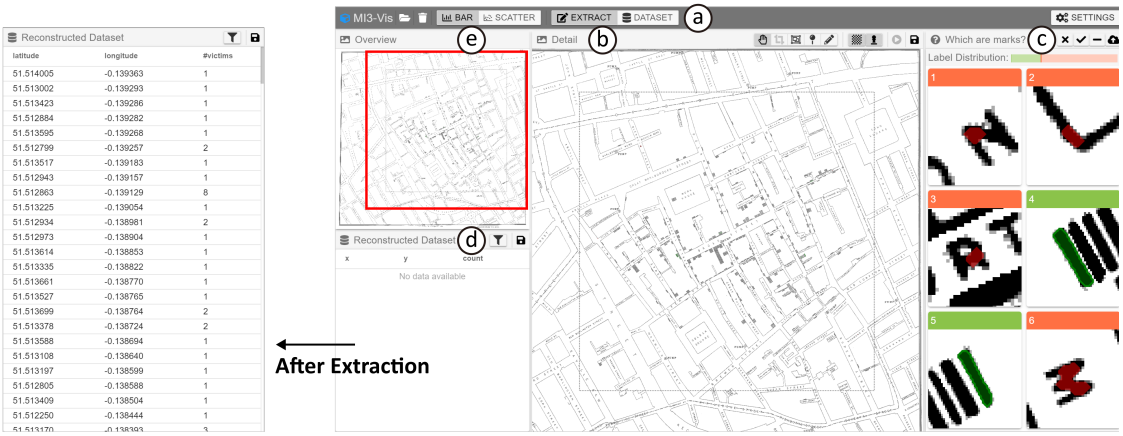


Figure 3.11: The prototype user interface for interactive classification and active learning. The interface consists of five parts: (a) Control Toolbar: the user can upload the image, select the chart type, switch interface layout, and change the algorithm parameters; (b) Image View: the Image View shows the input image and the re-visualized extracted dataset, and the user can navigate by pan and zoom, clip the image, set the scale of the image, manually create visual objects if needed, and conduct spatial filtering of data objects; (c) Annotation Panel: the user can annotate or correct the labels for data objects; (d) Dataset View: the Dataset View shows the extracted dataset as a table after the extraction process is finished; (e) Image Overview: the Image Overview highlights the currently zoomed region shown in the Image View.

menu for settings.

After the image is uploaded, the user can click the start button to start the data extraction. The user may clip the image to focus the algorithm on a subpart if needed. Once the extraction starts, the system goes through preprocessing, algorithmic sampling, and default labeling. The sampled objects in each iteration are displayed in the Annotation Panel (Fig. 3.11(c)). After the sampling, the Image View automatically zooms to the area where the sampled objects are located. The currently zoomed area is highlighted in the Image Overview (Fig. 3.11(e)). The user can pan and zoom the image in the Image View if needed. The Image View supports the manual creation of visual objects by drawing polygons, which provides a fallback option to the user when the preprocessing step of a data extraction workflow fails to recognize some of the visual objects.

In the Annotation Panel, the user can click on the thumbnail image to flip the label of a data object or interchangeably press the corresponding number key. The user can also click the object in the Image View to change the label. When the object's label is hard to determine from the thumbnail image and the user's mouse has been hovering on the thumbnail for more than 500ms, the system automatically zooms into a small area containing the corresponding object to help the user view the object more clearly.

The user can carry out quality assurance processes in this interface. In Chapter 5, we introduce an evaluation approach that involves modeling user operations in this interface.

Once the sampled data objects are checked, the user can click the confirm button in the Annotation Panel or press the "enter" key to add the data objects to the labeled set. The integrated interactive labeling and active learning process for one MI3 workflow runs iteratively until the termination criterion is met. After that, the process for the next MI3 workflow starts.

After all the MI3 workflows for data extraction are executed, the Dataset View (Fig. 3.11(d)) shows the extracted dataset. The user can export the extracted dataset in JSON format. The extracted dataset is also visualized and superimposed on the input visualization image. The extracted visualization can be exported in SVG format.

The user can optionally conduct coordinate transformation for the position attributes from the image's pixel coordinate to other coordinate systems. For coordinate transformation, the user needs to click the historical visualization image to pin three points and type their abscissas and ordinates in the transformed coordinate system, such as longitudes and latitudes. The system will then automatically update the position attributes in the Dataset View (Fig. 3.11(d)).

3.5 Evaluation with Simulations

In the previous section, we introduce multiple implementations of MI3 components that generate multiple instantiations of the MI3 workflow. For example, for the *algorithmic sampling* process, the developer may choose between multiple active learning strategies. This section evaluates the performance of different instantiations. We consider six major variations of solutions to the classification task:

- **BF** — The **brute force** approach where the user needs to use pen-and-ruler to measure and record all the data objects without the system's intelligent support.
- **ML + x%-PL** — A conventional **ML-based** system where the user **prelabels x**% of the training set. Then, a classification model is trained with the training set and assigns default labels for all the remaining data objects.
- **RS + RD + k-DL** — An instantiation of the MI3 workflow that exploits **random sampling**, and **random default labeling**. In each active learning session, **k** data objects are randomly sampled, randomly assigned a default label, and presented to the user. This provides a reference benchmark to evaluate the algorithmic sampling and default labeling techniques.
- **AS + RD + k-DL** — Similar to the above, except that **algorithmic sampling** is used in place of random sampling.
- **RS + AD + k-DL** — An instantiation of the MI3 workflow that exploits **random sampling**, and an **algorithmic default labeling** strategy. In each active learning

session, **k** data objects are randomly sampled and then default labeled according to the strategy. The number of user operations depends on the AD technique's error rate and the number k of data objects presented in each iteration.

- **AS + AD + k-DL** — Similar to the above, except that **algorithmic sampling** is used in place of random sampling.

Although it is useful to run user studies, the numerous system design options require many subjects and trials in the study, making it hard to carry out in practice. Simulation-based approaches are more feasible in such scenarios. Typical simulation-based evaluation methods include Card et al.'s keystroke-level model [48] and the later thread of work establishing the GOMS method [49, 136, 147]. In this section, to evaluate different options for instantiating the MI3 workflow, we adopt Zhang et al.'s simulation-based evaluation method [296]. Precisely, the number of user operations needed to finish the data extraction task serves as the evaluation metric and is estimated by the simulation. The simulation-based evaluation method makes it possible to gather many repeated trials, and the result does not suffer from the variance of human subjects.

The following assumes that the interaction time cost is proportional to the number of user operations. In Chapter 5, we refine the simulation-based evaluation approach by removing this assumption. Chapter 5 considers scenarios where the time costs of different types of user operations vary and need to be modeled individually.

3.5.1 Defining Ground Truth

Before examining data extraction accuracy, it is necessary to reflect on what "ground truth" and "accuracy" mean in the context of data extraction from historical visualizations. The following reflections inform our evaluation metric.

In various cases, assumptions must be made to generate the ground truth used for evaluation. While it is natural to suggest that the original numerical records the visualization authors' used should be the ground truth, the historical visualizations that we are especially interested in are the ones without such records. It is thus infeasible to validate data extraction "accuracy" against them.

Thus, the "ground truth" needs to be extracted from the historical visualization following a data extraction process. This the extracted "ground truth" is subject to various sources of errors, like any other data extraction process. The original author and publisher may make mistakes in crafting the historical visualization, and the printing technique may be imperfect. Both cases will leave insufficient information to recover the original data. The "ground truth" extraction process itself is subject to mistakes made by the person responsible for the data extraction.

As it is impossible to ensure the correctness of the "ground truth", we believe it sufficient in practice to give one's best guess when generating the "ground truth" for

evaluation. In our practice, we use the data we extracted ourselves as the "ground truth" for evaluation.

Additionally, the measurement of "accuracy" and the error tolerance depends on the data extraction task. For fields that encode real numbers, such as the location (x, y), it is hard and unnecessary to ensure all the digits of the extracted data are the same as the ones stored in the ground truth. Meanwhile, for the value $\#victims$, deviation from the ground truth is less tolerable.

In our scenario, as we convert data extraction tasks into classification tasks, we decide to use the classification accuracy of the converted task as the performance measure. Note that our evaluation metric ignores the sources of imprecisions that are not caused by misclassifications. For example, after all the data objects are extracted and their relations are detected, the user may transform the coordinates in pixels to longitudes and latitudes as introduced in Section 3.4.4. This process requires the user to type the longitudes and latitudes of three reference points, which may introduce imprecisions that the evaluation in this section does not cover.

3.5.2 Dataset

We use John Snow's cholera map [250] as the dataset for evaluation. In the cholera map, there are 579 rectangular blocks that form 321 bars. As illustrated in the previous sections, to extract data from this image, the user needs to carry out three classification tasks, i.e., classify candidate blocks, relations, and roots as true or false detections. For the block detection problem, there are 4416 candidate blocks to be classified detected by the preprocessing algorithm based on connected component detection. Among the 4416 candidates, 533 are positive (true detections), and 3883 are negative (false detections). Note that the preprocessing algorithm misses 46 blocks. Therefore, for this classification task, the user needs to make an additional effort to annotate these 46 blocks. For the neighboring relation classification problem, there are 868 candidate relations to be labeled where 258 are positive and 610 are negative. For the root classification problem, there are 450 candidate roots to be labeled where 321 are positive and 129 are negative. We generate the ground truth dataset is generated by first using our data extraction system and then manually verifying the extracted data.

We find some papers [152, 245] that suggest the cholera map contains 578 blocks (instead of 579 that we suggest), which *may* due to printing defect in the version of the cholera map they use. Dodson and Tobler's dataset[1] is the source that gives the number 578 [76]. After plotting Dodson and Tobler's dataset, we observe our counting

[1] https://github.com/vincentarelbundock/Rdatasets/blob/master/csv/HistData/Snow.deaths.csv hosts Dodson and Tobler's original dataset. Their original dataset contains two errors in block positions. https://github.com/vincentarelbundock/Rdatasets/blob/master/csv/HistData/Snow.deaths2.csv hosts a version of the dataset with the two errors fixed. In the main text, we refer to this dataset with errors fixed when talking about "Dodson and Tobler's dataset".

result (579) is different from theirs (578) because we count differently for the number of blocks located at the intersection of "NEW STREET" and "HUSBAND ST" in the map.

Dodson and Tobler used the version of the cholera map reprinted in 1936[2] as the data source [76]. In the 1936 version, there are seven blocks at the intersection of "NEW STREET" and "HUSBAND ST". In the version of the cholera map printed in 1855[3], and the most commonly used version of the cholera map on Wikipedia[4] that we use, there are eight blocks at the intersection. This difference between the cholera map versions leads to the difference in the counting result. We suspect the difference is caused by a printing defect in the 1936 version. In that version, one of the seven blocks is significantly larger than the others, implying that two blocks may appear as one large block because of imprecise printing.

3.5.3 Experiment Design

Compared Solutions: In the simulation, we compare the aforementioned six optional solutions to the classification tasks: (1) **BF**, (2) **ML + 20%-PL + NeD + 6-DL**, (3) **RS + RD + 6-DL**, (4) **AS + RD + 6-DL**, (5) **RS + AD + 6-DL**, and (6) **AS + AD + 6-DL**. Most of the data objects in the block detection dataset are false detections. For the **ML + x%-PL** solution using a conventional ML-based system with x% data objects prelabeled, we can enhance it by always prelabeling data objects to have negative default (NeD). Therefore, in the experiment, we substitute the conventional ML solution **ML + x%-PL** with **ML + x%-PL + NeD + k-DL**, which is a more competitive benchmark. We set the prelabel rate x% to be 20%. For all the solutions, we fix the number of data objects k presented in each active learning session to be 6. For conciseness of the result, we fix entropy-based sampling to be the instantiation of active sampling (AS (ES)) and the interim decision tree model to be the model for algorithmic default labeling (AD (Interim)).

Evaluation Metric: As we have converted data extraction to classification tasks, we use the accuracy of the classification as the evaluation metric. In the experiment, for each of the three MI3 workflows, we measure the user operation cost for each optional solution to achieve 100% accuracy in classification. We consider four types of user operations in the measurement: (1) a button clicking in the Annotation Panel (Fig. 3.11(c)) to edit the label of a single data object, (2) a button clicking in the Annotation Panel (Fig. 3.11(c)) to confirm the label of a batch of data objects, (3) a polygon creation in the Image View (Fig. 3.11(b)) to extract a missing data object in the preprocessing, and (4) a button clicking in the Image View (Fig. 3.11(b)) to correct mislabeled data objects. For example, in the block detection problem, there are 579 true blocks. The preprocessing detects 4416 data objects to be classified,

[2]The map is available at https://archive.org/details/b3134818x_0001/page/46
[3]The map is available at https://archive.org/details/b28985266/page/n57
[4]The map is available at https://en.wikipedia.org/wiki/File:Snow-cholera-map-1.jpg

where 533 are true detections. For the **BF** solution using pen-and-ruler or its digital equivalent, it would take a minimal 579 user operations to measure and record all the data objects. For the conventional ML solution with **ML + 20%-PL + NeD + 6-DL**, it would require $883 \approx 4416 \times 20\%$ data objects to be labeled in the prelabeling session. Approximately 12% of the data objects are true detections. Therefore, the negative default strategy (NeD) would produce incorrect default labels for 12% of the 883 data objects. It needs $108 \approx 883 \times 12\%$ user operations for correcting incorrect default labels and $148 \approx 883/6$ user operations to click the confirm button to register the corrections. An additional 46 user operations are needed to correct missing blocks in the preprocessing. Our initial testing result shows that with 883 data objects labeled, the model achieved 98.837% accuracy for the remaining $3533 = 4416 - 883$ data objects. Therefore, $41 \approx (1 - 98.837\%) \times 3533$ user operations are needed to correct the misclassifications. In total, the number of user operations to achieve 100% accuracy for **ML + 20%-PL + NeD + 6-DL** is $343 = 108 + 148 + 46 + 41$. Note that for simplicity, when counting the number of user operations, we have assumed the user does not make mistakes in carrying out the operations. When user mistakes are considered, the number of user operations may increase as the user may need additional operations to correct mistakes.

We can estimate the interaction cost for other optional solutions similarly. Note that for the four instantiations of MI3, the interaction cost depends on the number of active learning sessions conducted. With more sessions, more user operations are needed for default label correction. Meanwhile, the more data objects are labeled, the higher the model's accuracy, which reduces the interaction cost to quality-assure and correct the labels for the remaining data objects. To show these dynamics, the overall interaction cost can be represented as a function of the number of user operations in active learning sessions. The number of user operations in active learning sessions excludes the interaction cost for the final quality assurance and correction of the remaining data objects' labels. For the block detection problem, we run 75 repeated measures and average the results. For the neighboring relation detection and root detection problems, we run 150 repeated measures and average the results.

We implement the simulation function to loop over the dataset batch by batch and log the sequence of user operations needed to accomplish a data extraction task. As mentioned above, when logging the user operations, we assume: (1) the user makes no mistakes. Additionally, in the following, we use the number of user mouse operations as a proxy of cost, which implies the following assumptions: (2) only mouse operations have costs (Chapter 5 relaxes this assumption), and (3) each type of user operation costs the same (Chapter 5 relaxes this assumption). Under these assumptions, the total number of user mouse operations captures the total cost.

3.5.4 Results

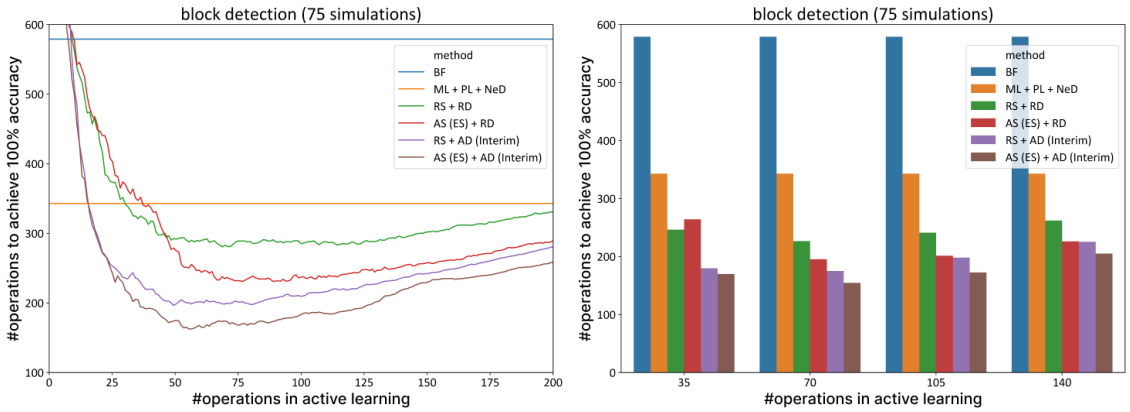


Figure 3.12: Comparison of the 6 optional solutions to the block detection problem. The curves on the left show the change of interaction cost to achieve 100% accuracy with regard to the interaction cost in active learning. The bar chart on the right compares the performance of the optional solutions at 35, 70, 105, and 140 user operations for active learning. The correspondence between solutions and colored curves and bars is: blue for the brute force approach (**BF**); orange for the conventional ML-based approach with 20% data objects prelabeled and negative default labeling (**ML + 20%-PL + NeD + 6-DL**); green for random sampling with random default labeling (**RS + RD + 6-DL**); red for algorithmic sampling with random default labeling (**AS + RD + 6-DL**); purple for random sampling with algorithmic default labeling (**RS + AD + 6-DL**); brown for algorithmic sampling with algorithmic default labeling (**AS + AD + 6-DL**). All the solutions except **BF** show 6 data objects in each labeling session, and therefore, the "**6-DL**" is omitted in the legend.

Fig. 3.12 shows the experiment results in block detection with the six optional solutions. In the following, we discuss patterns observed in the experiment result. For conciseness, we refer to the total interaction cost to finish the classification task (i.e., achieve 100% accuracy) to be the "overall cost", the interaction cost in the iterative active learning sessions to correct the data objects with incorrect default labels to be the "active learning cost", and the interaction cost in the final quality assurance session to correct mislabeled data objects to be "quality assurance cost".

1. **Performance of MI3 workflow instances follows a U-shaped curve.** The overall costs of **RS + RD**, **AS + RD**, **RS + AD**, and **AS + AD** follow the same trend of first decrease and then increase with the active learning cost. Such a pattern emerges because the overall cost is comprised of the active learning cost and quality assurance cost. The quality assurance cost typically experiences a rapid decrease during the first few active learning sessions, as the accuracy of a classifier typically improves faster with the increase of training data at the beginning. With more user operations spent on active learning, the rate that quality assurance cost decreases gets slower as the interim classification model's performance reaches a plateau. At some point, the increased contribution of active learning

cost to the overall cost would overcome the decrease of the quality assurance cost, making the overall cost increase again.

Because of the U-shaped pattern of the overall cost, for each optional solution, there is an optimal point of the total number of user operations for active learning that minimizes the overall cost. For example, for **RS + RD**, the optimal point is when 68 user operations are spent on active learning. The overall cost at the optimal point depicts the performance of the solution when the iterative labeling process terminates at a suitable time point, as determined by the termination criterion. Assuming that the active learning sessions terminate at suitable time points, we compare different solutions mainly by the lowest point of the curve in the following three observations.

2. **MI3 workflow instances outperform the brute force approach and the conventional machine learning workflow.** The optimal overall costs of **RS + RD**, **AS + RD**, **RS + AD**, and **AS + AD** are smaller than the overall costs of **BF** and **ML + PL + NeD**. For the block detection task, it takes **BF** 579 user operations to record all the blocks. It takes **ML + PL + NeD** 343 user operations to detect all the blocks with 100% accuracy. By comparison, the four MI3 instances have optimal overall costs between [150,300] user operations.

3. **Algorithmic sampling (AS) reduces interaction cost.** For MI3 instances with the same default labeling strategy, the instance using an algorithmic sampling strategy outperforms the instance using random sampling. Specifically, the overall cost of **RS + RD** reaches its minimum, 280 user operations, with 68 user operations for active learning, and then gradually increases. When the sampling method is fixed as **RD**, after incorporating algorithmic sampling, **RS + RD** transforms to **AS + RD**. The optimal overall cost reduces to 230 user operations, which is achieved at 76 user operations for active learning. Similar differences are observed between **RS + AD** and **AS + AD**. We interpret the smaller overall interaction cost as algorithmic sampling manages to continuously explore informative samples for the interim model to learn, which sustainably decreases the number of misclassifications and reduces the interaction cost to correct.

4. **Algorithmic default labeling (AD) reduces interaction cost.** For MI3 instances with the same sampling strategy, the instance using an algorithmic default labeling strategy outperforms the instance using random default labeling. With algorithmic default labeling, a lower overall cost can be achieved with the same number of user operations compared with random default labeling. The overall cost of **RS + RD** at the optimal point is 280 user operations and is larger than that of **RS + AD**, which is 196 user operations. This is because algorithmic default labeling is typically more accurate than random default. Therefore, algorithmic

default labeling enables more data objects to be labeled with the same active learning cost. Thus, the quality assurance cost of algorithmic default labeling is less than that of random default labeling when their active learning costs are the same.

In summary, instantiations of the MI3 workflow outperform the brute force approach (**BF**) and the conventional machine learning workflow (**ML + 20%-PL + NeD + 6-DL**). Moreover, among different instantiations of the MI3 workflow, we find that for the sampling component, workflows using algorithmic sampling (**AS**) perform better than those using random sampling (**RS**). For the default labeling component, workflows using algorithmic default labeling (**AD**) are better than those using random default labeling (**RD**). Among the implemented MI3 instances, the one with the best performance, **AS + AD**, requires a minimal 162 user operations to accomplish the classification task with 100% accuracy, which is less than one-third the cost of the brute force approach, and less than half the cost of the conventional machine learning workflow.

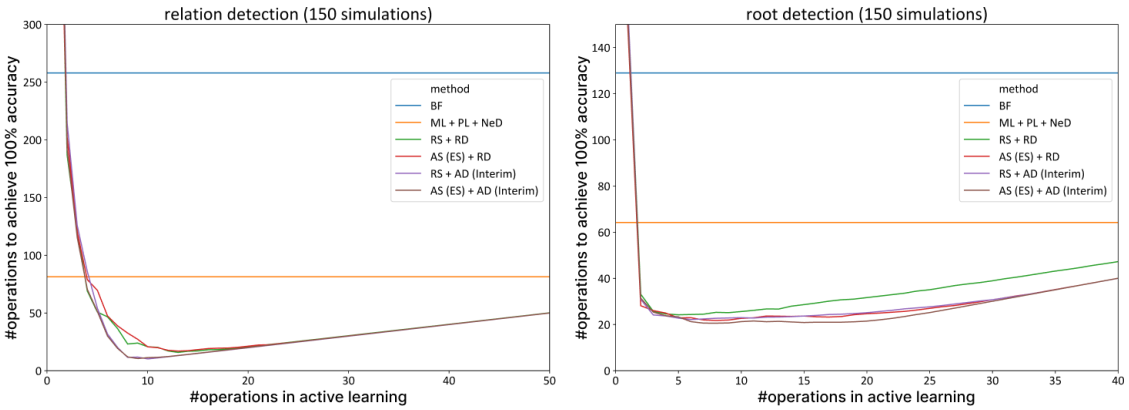


Figure 3.13: Comparison of the 6 optional solutions in neighboring relation detection and root detection. The curves show the change of overall interaction costs with regard to the interaction cost in active learning.

Aside from the block detection problem, we also evaluate the six workflows on neighboring relation detection (for block grouping) and root detection. The experiment results are shown in Fig. 3.13. We observe that similar to the result of block detection, MI3 workflow instances outperform the brute force approach and the conventional machine learning workflow. However, for these two subtasks, algorithmic sampling (**AS**) and algorithmic default labeling (**AD**) bring insignificant benefits over random sampling (**RS**) and random default labeling (**RD**). We interpret it as the result that the two subtasks are much simpler than the block detection problem. Both relation detection and root detection can be accomplished with less than 30 user operations.

3.6 Analyzing Reusable Workflows for Data Extraction

Although the above workflows based on active learning can be more efficient and effective than the conventional machine learning pipeline, there is a development cost associated with each pipeline. Unless each pipeline can be reused for many historical visualization images, the development cost can still be significantly higher than measuring every data object manually and collecting the measured data using a spreadsheet.

We surveyed some ten archive sources, such as the Library of Congress [170], United States Census Records [263], David Rumsey Map Collection [231], Old Maps Online [151], Milestones Project [94], and the Historical Atlas of China [254]. There are estimated at least 500,000 historical visualizations in various collections worldwide. While most of them are geographical maps, there are also many statistical graphics, thematic maps, and illustrations.

For the MI3 approach to be developed into a software system that can be applied to most of these visualizations, the number of active learning workflows required must be significantly smaller than 500,000. If the number of workflows is moderate, not only is it more cost-beneficial to develop the software system but also likely it is easy and quick for a user to configure a multi-workflow pipeline for a given individual visualization without the need for any programming effort. This section discusses the feasibility of such a software system by examining the reusability of MI3 workflows. We demonstrate that a small collection of data extraction workflows can be assembled into data extraction pipelines for various scenarios. We give two additional examples of building data extraction systems for line charts and Sankey diagrams by composing reusable workflows.

Fig. 3.14 shows nine examples of historical visualizations. They cover some commonly used visualization designs nowadays, such as line charts, bar charts, scatterplots, node-link diagrams, and thematic maps, some with embedded glyphs. With a quick observation, one can easily see that our three workflows in Section 3.4 can be reused for several examples. For instance, the block detection workflow can possibly be used to detect bars in (b) and (c), circles and wedges in (d) and (e), nodes and edges in (f), and dots in (h). The block grouping workflow can possibly be used to group the bars in a bar-chart glyph in (c) and the wedges in a pie-chart glyph in (e). The root detection workflow can possibly be used to detect the top-middle location of each bar in (b) and (c), the center-bottom position of each bar-chart glyph in (c), the circular center of each bubble in (d) and each pie-chart glyph in (e), and the center of each node in (f) and each dot in (h).

While each workflow exhibits reasonable reusability, more workflows will be needed to transform the MI3 approach into a practical software system. After analyzing over 150 historical visualization images in detail and conceptually designing a vocabulary

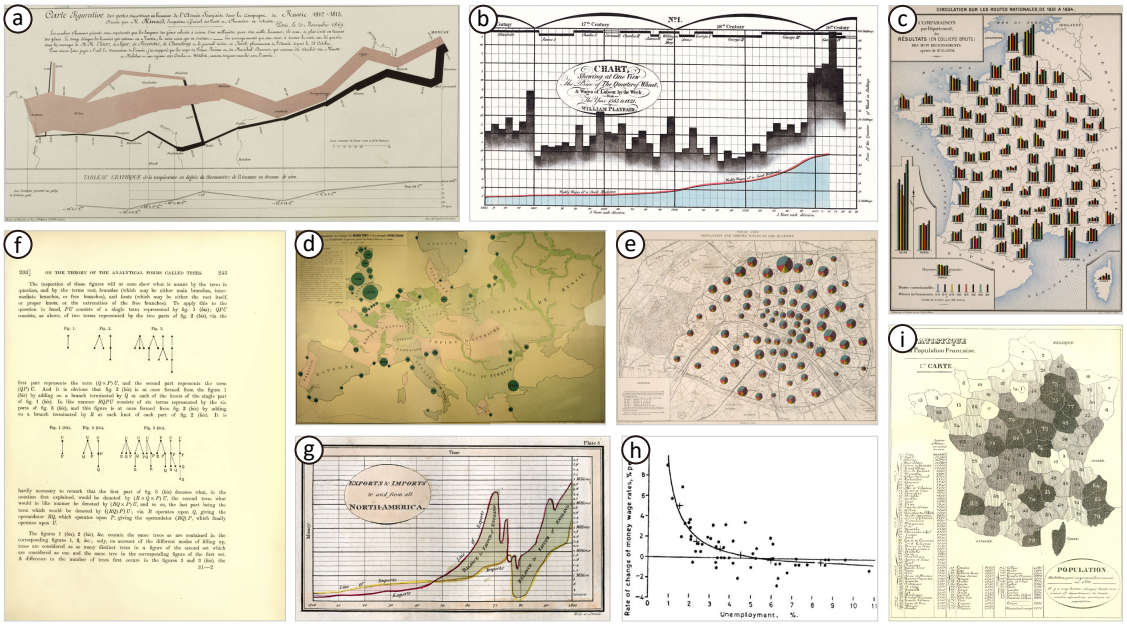


Figure 3.14: Examples of historical visualizations: (a) Charles Minard's movement flow map, 1869 [190]; (b) William Playfair's bar chart, 1822 [214]; (c) Émile Cheysson's thematic map with bar-chart glyphs, 1906 [59]; (d) Charles Minard's bubble map, 1859 [189]; (e) Jacques Bertillon's thematic map with pie-chart glyphs, 1886 [27]; (f) Arthur Cayley's node-link diagram, 1890 [50]; (g) William Playfair's line chart, 1784 [209]; (h) William Phillips's scatterplot, 1958 [208]; (i) Adolphe d'Angeville's choropleth map, 1836 [70].

for extracting data from each of the images, we identified the following list of 18 workflows as building blocks that can be used in combination to support the data extraction from numerous historical visualization images.

3.6.1 Reusable Workflows

Visual Object Detection

a_1. *Component-based object classification* — given a collection of imagery components, determine if a component is a visual object of a certain type (e.g., blocks in the cholera map, dots in a scatterplot, or nodes in a graph), *while learning a model to do this semi-automatically.* The italicized clause applies to all the following active learning workflows and is omitted.

a_2. *Seed-based object construction* — given a seed on an object, determine all the pixels that belong to the same object (e.g., searching for a region in a map).

a_3. *Scan-based object detection* — given an image, examine a small area of the image by following a search path and determine if an object is in the area or partially included in the area. It is usually suitable for objects of a fixed size (e.g., a dot in a scatterplot, a node in a graph, or a character in a text string). It can be used

to detect seeds before invoking workflow a_2 or a starting object before invoking workflow a_4.

a_4. *Tracking-based object detection* — given an image and a set of previously tracked objects, determine all pixels that belong to the next object, which may be a group of pixels along a path (e.g., a marker in a line chart), a line of pixels orthogonal to the path or an axis (e.g., a cross-section in Charles Minard's flow map [190] or a Sankey diagram).

Visual Object Organization

b_1. *Relation detection and classification* — given two or more visual objects, determine whether they are related and of what type of relation (e.g., blocks forming a bar in the cholera map, two nodes of an edge in a graph, or wedges forming a pie chart).

b_2. *Object ordering* — given a set of visual objects, produce an ordered list.

b_3. *Object grouping* — given two or more visual objects and a set of relations among them (e.g., detected using b_1, construct a group object with these objects or a subset of them). In many cases, only a simple grouping procedure is needed, which requires little active learning.

b_4. *Object segmentation* — given an imagery area or a visual object, divide the area or the object into sub-objects (e.g., wedges in a pie chart following circle detection).

Visual Variable Measurement

c_1. *Position measurement* — given a visual object or a group of visual objects, determine a position (x, y) (e.g., the root of a bar in the cholera map, or the center of a pie chart).

c_2. *Length measurement* — given a visual object or a group of visual objects, determine a value $length$ that represents a length measure (e.g., width, height, one of the two diagonal lengths, maximal/minimal distance between points on the contour, cross-section length, path length, or arc length).

c_3. *Area measurement* — given a visual object or a group of visual objects, determine a value $area$ that represents an area measure (e.g., bounding box area, bounding circle area, foreground area, inner circle area, or the area with a specific subset of colors).

c_4. *Angle measurement* — given a visual object or a group of visual objects, determine a value $angle$ that is measured for a wedge or curve featured in the object or group.

c_5. *Direction measurement* — given a visual object or a group of visual objects, determine a normalized or unnormalized vector (dx, dy) that is measured for the object or group.

c_6. *Scale measurement* — given a visual object or a group of visual objects, determine a mapping $scale$ for the color intensity, hue-specific color intensity, texture density, color saturation, color hue, or other options.

c_7. *Group measurement* — given a group of objects, determine a value $value$ that represents the number of visual objects in the group, or the sum, average, maximum, or minimum of the values associated with the visual objects in the group.

c_8. *Measurement transformation* — given the measured value(s) of a visual object or a group of visual objects, determine a function for transforming the visual variable(s) to data variable(s) (e.g., scaling according to an axis, applying the mapping specified by a size legend, applying logarithmic factor, applying power factor, or percentage normalization).

c_9. *Number recognition* — given a visual object or a group of visual objects, determine a value $value$ that is an integer or a real number.

c_{10}. *Text recognition* — given a visual object or a group of visual objects, determine a value $text$ that is a character or a string of characters.

3.6.2 Composing Workflows

The three workflows described in Chapter 3 are a_1 (Section 3.4.1), b_1 (Section 3.4.2), and c_1 (Section 3.4.3). Tables 3.1 and 3.2 show some multi-workflow pipelines for extracting data from the nine historical visualizations in Fig. 3.14. All of these pipelines can be configured with the above 18 active learning workflows, demonstrating the potential reusability of a relatively small set of workflows. As shown in the tables, there can be multiple ways to extract data from a visualization using the vocabulary of workflows.

A workflow can be implemented as an interactive machine learning process as introduced in Section 3.2. It may also be implemented as automatic algorithms or purely manual operations. We anticipate two kinds of situations where it may be more cost-effective for a user not to use interactive machine learning for a task.

In the first kind of situation, although a workflow may offer a number of algorithmic solutions or various optional parameter settings of an algorithm, the user knows exactly which algorithmic solution or which parameter setting to use. It is more cost-effective for a user to choose the algorithm or parameter setting interactively, e.g., through a pop-up window. For example, the workflow c_1 position measurement has the flexibility to measure the position (x, y) of a visual object by learning to recognize the signature

Table 3.1: Optional pipelines for extracting data in Fig. 3.14(a - d), which may be configured using the 18 active learning workflows described earlier in this section. Each pipeline is denoted as $\mathbf{W}_{\text{datatype},k}$, where k indicates the k-th option. When a workflow is given a superscript "m", "a", or "ma", it is likely most cost-effective for the workflow to activate a purely manual, a simple algorithmic, or a parameterized algorithmic process instead of active learning. Almost all pipelines will require the workflow c_8 (measurement transformation), and thus it is not shown explicitly in the table.

Example	Pipelines
Fig. 3.14(a) *Movement flow map and line chart*	$\mathbf{W}_{\text{flow},1}$: a_3^m (start cross-section) $\rightarrow c_1^a$ (position) $\rightarrow c_2$ (width) $\rightarrow [\![a_1$ (stop detection (branch/merge/sharp-turn)) $\rightarrow a_4$ (next cross-section) $\rightarrow c_1^a$ (position) $\rightarrow c_2$ (width)$]\!] \rightarrow b_3$ (grouping all the flow segments) $\mathbf{W}_{\text{flow},2}$: a_1 (flow segment) $\rightarrow b_4$ (flow segment to cross-section line segments) $\rightarrow c_2^a$ (width) $\rightarrow c_1^a$ (position) $\rightarrow b_3$ (linking all the line segments) $\mathbf{W}_{\text{line},1}$: a_3^m (start point) $\rightarrow c_1^a$ (position) $\rightarrow [\![a_4$ (next point) $\rightarrow c_1^a$ (position)$]\!]$
Fig. 3.14(b) *Bar chart and line chart*	$\mathbf{W}_{\text{bar},1}$: a_3^m (start point of the top-left of the first bar) $\rightarrow c_1^a$ (position) $\rightarrow [\![a_4$ (next point along the staircase line) $\rightarrow c_1^a$ (position)$]\!] \rightarrow b_3$ (points to line segments) $\rightarrow c_1^a$ (position) $\mathbf{W}_{\text{bar},2}$: a_3^{ma} (seed per bar) $\rightarrow [\![a_4^a$ (tracking up) $\rightarrow c_1^a$ (position)$]\!] \rightarrow c_8^a$ (uniformly redistributing points horizontally) $\mathbf{W}_{\text{line},1}$: a_3^m (start point) $\rightarrow c_1^a$ (position) $\rightarrow [\![a_4$ (next point) $\rightarrow c_1^a$ (position)$]\!]$ $\mathbf{W}_{\text{line},2}$: $[\![a_1$ (line segment) $\rightarrow b_4^a$ (segment to points) $\rightarrow c_1^a$ (position)$]\!] \rightarrow b_2^a$ (point ordering) $\mathbf{W}_{\text{line},3}$: a_1 (line segment) $\rightarrow b_1^a$ (segment grouping) $\rightarrow b_4^a$ (line to points) $\rightarrow c_1^a$ (position) $\mathbf{W}_{\text{line},4}$: a_1 (cyan region) $\rightarrow b_4^a$ (region to vertical cross-sections) $\rightarrow c_2^a$ (cross-section length) $\mathbf{W}_{\text{line},5}$: a_3^m (seed) $\rightarrow a_2$ (cyan region) $\rightarrow b_4^a$ (region to vertical cross-sections) $\rightarrow c_2^a$ (cross-section length)
Fig. 3.14(c) *Thematic map with bar-chart glyphs*	$\mathbf{W}_{\text{bar},1}$: a_1 (bar) $\rightarrow c_2^a$ (bar height) $\rightarrow b_1$ (bar relation) $\rightarrow b_3^a$ (bar grouping) $\rightarrow b_2^a$ (bar ordering) $\rightarrow c_1$ (glyph position) $\mathbf{W}_{\text{bar},2}$: a_1 (glyph) $\rightarrow c_1$ (glyph position) $\rightarrow b_4$ (glyph to bars) $\rightarrow c_2^a$ (bar height) $\rightarrow b_2^a$ (bar ordering)
Fig. 3.14(d) *Bubble map*	$\mathbf{W}_{\text{bubble},1}$: a_1 (bubble) $\rightarrow c_3^a$ (bubble area) $\rightarrow c_1^a$ (bubble position) $\mathbf{W}_{\text{bubble},2}$: a_1 (patch with number) $\rightarrow c_9$ (number) $\rightarrow c_1$ (number position)

Table 3.2: Optional pipelines for extracting data in Fig. 3.14(e - i), which may be configured using the 18 active learning workflows described earlier in this section. See Table 3.1 for notational information.

Example	**Pipelines**
Fig. 3.14(e) *Thematic map with pie-chart glyphs*	$\mathbf{W}_{\text{glyph,1}}$: a_1 (wedge) $\rightarrow$ c_3^a (wedge area) $\rightarrow$ b_1 (wedge relation) $\rightarrow$ b_3^a (wedge grouping) $\rightarrow$ c_1^a (circle position) $\rightarrow$ c_7^a (circle area) $\rightarrow$ c_8^a (percentage normalization) $\mathbf{W}_{\text{glyph,2}}$: a_1 (circle) $\rightarrow$ c_1^a (circle position) $\rightarrow$ c_3^a (circle area) $\rightarrow$ b_4 (wedge segmentation) $\rightarrow$ c_3^a (wedge area) $\rightarrow$ c_8^a (percentage normalization) $\mathbf{W}_{\text{glyph,3}}$: a_3 (seed in circle) $\rightarrow$ a_2 (circle) $\rightarrow$ c_1^a (circle position) $\rightarrow$ c_3^a (circle area) $\rightarrow$ b_4 (wedge segmentation) $\rightarrow$ c_3^a (wedge area) $\rightarrow$ c_8^a (percentage normalization)
Fig. 3.14(f) *Node-link diagram*	$\mathbf{W}_{\text{graph,bottom}}$: a_3 (node) $\rightarrow$ a_3 (label) $\rightarrow$ c_{10} (recognize label) $\rightarrow$ b_1 (node with node label) $\rightarrow$ a_4 (edge) $\rightarrow$ b_1 (source node per edge) $\rightarrow$ b_1 (destination node per edge) $\mathbf{W}_{\text{graph,top}}$: same as $\mathbf{W}_{\text{graph,bottom}}$ except without c_{10} (recognize label)
Fig. 3.14(g) *Line chart*	$\mathbf{W}_{\text{line,1}}$: a_3^m (red line start point) $\rightarrow$ c_1^a (position) $\rightarrow$ $[\![a_4$ (next point) $\rightarrow$ c_1^a (position)$]\!]$ $\rightarrow$ a_3^m (yellow line start point) $\rightarrow$ c_1^a (position) $\rightarrow$ $[\![a_4$ (next point) $\rightarrow$ c_1^a (position)$]\!]$
Fig. 3.14(h) *Scatterplot and line chart*	$\mathbf{W}_{\text{point,1}}$: a_1 (dot) $\rightarrow$ c_1^a (dot position) $\mathbf{W}_{\text{line,1}}$: a_3^m (start point) $\rightarrow$ c_1^a (position) $\rightarrow$ $[\![a_4$ (next point) $\rightarrow$ c_1^a (position)$]\!]$
Fig. 3.14(i) *Choropleth map*	$\mathbf{W}_{\text{region,1}}$: a_3 (seed) $\rightarrow$ a_2 (region) $\rightarrow$ c_6 (texture density) $\rightarrow$ c_9 (number) $\rightarrow$ b_4^a (region to boundary points) $\rightarrow$ c_1^a (point position) $\mathbf{W}_{\text{region,2}}$: a_1 (border curve segments) $\rightarrow$ b_4 (curve segment to points) $\rightarrow$ c_1^a (point position) $\rightarrow$ b_1 (curve relation) $\rightarrow$ b_3 (curves to region) $\rightarrow$ c_6 (texture density) $\rightarrow$ c_9 (number)

of the position, such as top-middle, bottom-right, the center of a bounding circle, or a red dot on the object. Active learning is necessary for position measurement in John Snow's cholera map. The notion of "bottom block" is not hard-coded as a predefined signature. However, for the bubbles in Fig. 3.14(d), the circles in (e), and the dots in (h), the user can select the predefined signature "center" in a pop-up window for c_1 when configuring the corresponding pipeline. For such situations, we add a superscript "a" (for algorithmic) to the label of the workflow concerned.

In the second kind of situation, there may not be enough visual objects for a workflow to learn a model that can offer any assistance. For example, there is only one starting cross-section for the big flow segment in peach color in Fig. 3.14(a), and only one starting point for the line chart in Fig. 3.14(a), (b), and (h). It will be much quicker for a user to specify such a starting object manually. The option of manual

specifications can be made available through a pop-up window for configuring the corresponding workflow a_3 (i.e., scan-based object detection). For such situations, we add a superscript "m" (for manual) to the label of the workflow concerned.

In some cases, a user may identify a sequence of built-in algorithms and configure an automatic or semi-automatic pipeline without the need for active learning, such as $\mathbf{W}_{\text{bar},2}$ for Fig. 3.14(b). In other cases, although one or a few active learning workflows are used in conjunction with built-in algorithms, the active learning becomes more or less a manual labeling process because there are not enough visual objects. $\mathbf{W}_{\text{flow},2}$ for Fig. 3.14(a) is such an example.

In many ways, these 18 workflows are building blocks of most pipelines for data extraction from historical visualizations. In a practical system, there may be a few more or fewer workflows. Nevertheless, it is unlikely that it is necessary to have many more workflows. It is unlikely that in the near future, an AI system can scan an arbitrary historical visualization and work out what are the visual objects that depict the required data and what may be the appropriate workflows for the data extraction task. By allowing users to configure a pipeline for a data extraction task, we enable users to use their knowledge about the semantics of a visualization and their intelligence to select a workflow for the task. Abdul-Rahman et al. presented a visual analytics system for text similarity detection [2], which enabled users to configure a detection workflow by selecting various algorithmic modules and setting appropriate parameters for the modules. Their system provides 41 algorithmic modules and allows three parallel workflows. They reported that humanities scholars were able to learn how to configure such a workflow after two hours of training. Based on their experience, we infer that it is feasible for users to configure a pipeline with 18 optional workflows.

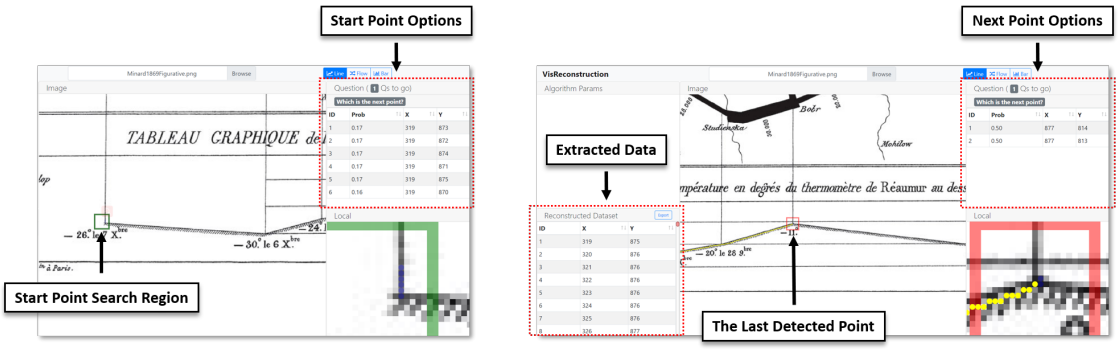
3.6.3 Prototypes

To demonstrate the feasibility of the data extraction pipelines designed above, the following sketches prototype implementations of two of the pipelines for line chart ($\mathbf{W}_{\text{line},1}$) and Sankey diagram ($\mathbf{W}_{\text{flow},2}$). We believe that the remaining data extraction pipeline in Tables 3.1 and 3.2 are also feasible and can be implemented in the future.

Line Chart Data Extraction

For line chart data extraction, the goal is to extract the dataset encoded in the position attribute of the points along the line: $(x_1, y_1), ..., (x_n, y_n)$. Table 3.1 $\mathbf{W}_{\text{line},1}$ decomposes the task into three components:

1. **Start Point Detection** (a_3): Find the start point of a line.
2. **Next Point Tracking** (a_4): Iteratively find the next point of the line, given the points already detected on the line.



(a) Start Point Detection (b) Next Point Tracking

Figure 3.15: The interface of the prototype line chart data extraction system. (a) **Start Point Detection**: The user clicks on the canvas to specify a coarse region of the start point and chooses the exact start point among the listed options. (b) **Next Point Tracking**: The user chooses the next point among candidates suggested by the algorithm.

3. **Position Measurement** (c_1): Measure the positions of points on the line.

Position measurement can be implemented as measuring the points' position in the image and optionally going through a scale transformation. Thus, the following focuses on start point detection and next point tracking.

Start Point Detection We blur the image with a gaussian filter, binarize the image, and detect skeletons by morphological thinning [110]. The pixels on the skeletons are regarded as candidate line chart pixels. Thus, detecting line chart pixels converts to classifying skeleton pixels. We design a semi-automatic process. The user first coarsely selects a square region containing the start point. Assuming the line chart is horizontal, we filter the candidate pixels with two heuristic rules. Firstly, the start point should be connected to a foreground pixel on the selected square region's top, bottom, or right border. Secondly, on the left of the start point, there should be no foreground pixel 8-connected to it. The user can then choose the start point among the candidates, as shown in Fig. 3.15(a).

Next Point Tracking The iterative tracking of the next points can be implemented as an interactive machine learning process. The user labels the candidates for the next points, and the algorithm iteratively learns to track the next points with gradually reduced user interventions. Specifically, given the last detected point on the line, the pixels in its neighborhood are regarded as the possible next point. The task is then to classify which candidate pixel is the next point on the line. The interface provides candidates to the user, as shown in Fig. 3.15(b). For each candidate option, we can compute features, including position differences, smoothness, and color differences. The system learns a classifier through the user's choices. When the classifier's confidence is higher than a threshold, the tracking is conducted automatically without user intervention.

Sankey Diagram Data Extraction

For Sankey diagram data extraction, the goal is to extract the dataset that can be represented as a directed graph. A node can be stored as $(x, y, w, sources, targets)$ where (x, y) is the node position, and w is the flow width at the node. When the flow is not starting, ending, branching, or merging at (x, y), its source list $sources$ and target list $targets$ point to one node. When the flow merges/branches at (x, y), its source/target list points to multiple nodes. Table 3.1 $\mathbf{W}_{\text{flow},2}$ decomposes the task into five steps:

1. **Flow Segmentation** (a_1): Extract connected components belonging to flows.
2. **Medial Axis Extraction** (b_4): Extract the medial axes of each flow and detect the branch and merge points.
3. **Width Measurement** (c_2): Measure the encoded value at each point along the medial axes by the flow width.
4. **Position Measurement** (c_1): Measure the positions of points along the medial axes.
5. **Line Segment Linking** (b_3): Link the lines corresponding to the same flow.

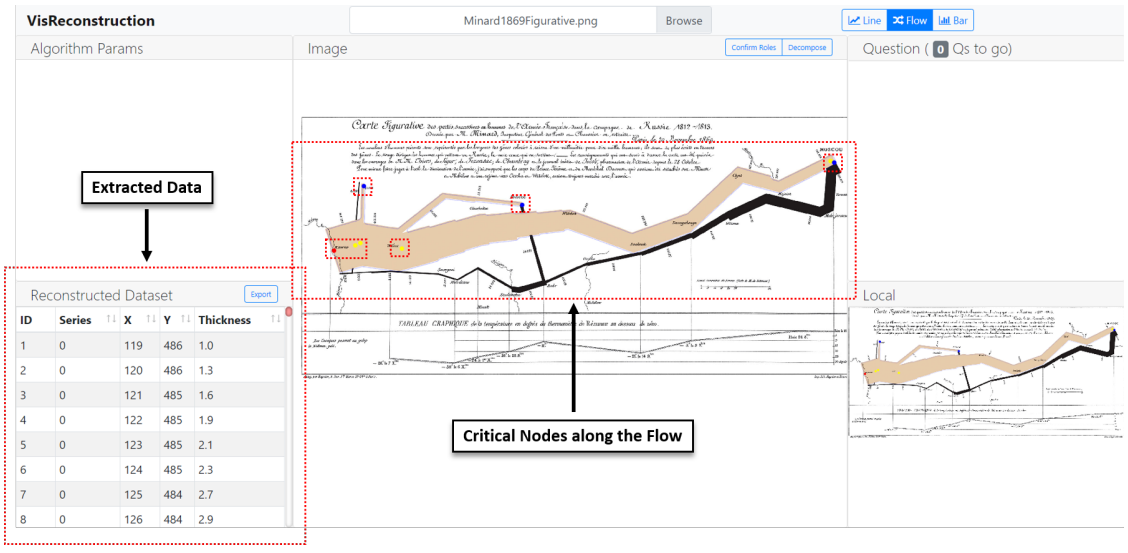


Figure 3.16: The interface of the prototype Sankey diagram data extraction system.

Like the line chart, measuring width and position is straightforward. The following focuses on flow segmentation, medial axis extraction, and line segment linking.

Flow Segmentation Given a Sankey diagram image, the gaussian filter is used for smoothing. Then, the user needs to click on the flow component. The machine then detects connected components with a similar color to the clicked point.

Medial Axis Extraction The system detects the skeletons by morphological thinning [110] and builds a graph. The nodes store the pixels at the ends and joint points of the skeleton. The edges store the line segments on the skeleton between the nodes. The roles of the nodes are classified heuristically as starting, ending, branching, and merging nodes. The left-most node is assumed to be the start point and the other nodes at the ends as the end point. The user can correct the heuristic role classification, as shown in Fig. 3.16.

Line Segment Linking A flow may appear segmented because of occlusion by other visual objects. For example, the flow colored black in Minard's map corresponds to three segments. It is necessary to link these segments and interpolate the data of the occluded part. The linking can be formulated as classifying whether to link a pair of end nodes. The classification can account for features including the distance between the points, the difference in tangent line angles, and the distance to the intersecting point.

3.7 Domain Expert Interview

To examine if the data extraction systems can be helpful to historians, we carry out informal interviews to gather domain experts' comments.

3.7.1 Interview Design

Participants: From three universities, we found four historians as voluntary participants. They had history study and research experience in universities between 4 to 7 years. P1 studied East Asian history and warfare history. P2, P3, and P4 studied Chinese history with different focuses. P2 studied classical philology and socioeconomic history. P3 studied the social history of Qin, Han, and the Six Dynasties. P4 studied new cultural history.
Procedure: The interview lasted between an hour to two hours for each participant. Through the interview, we aim to learn about how the participant previously used historical visualizations in their research and how data extraction can be helpful to them. We carry out unstructured interviews with the following question prompts:

Q1. How many years have you studied and researched history?

Q2. What is your research interest?

Q3. Have you used historical visualizations for your research?

Q4. Do you think the data extracted from historical visualizations are useful to you?

Q5. How would you use the extracted data?

Q6. What do you think about our systems for data extraction?

Q1 and Q2 concern the participants' backgrounds. Q3, Q4, and Q5 concern the participants' previous and potential plans of using historical visualizations. Q6 concerns the usefulness and usability of our systems. In the interviews, we raised questions to the participants following in the order of Q1 to Q6. In addition to the question prompts, we ask additional questions when necessary. Between Q3 and Q4, we demonstrate to the participant the functionalities of the data extraction system of Chapter 3 and the authoring toolkit of Chapter 4.

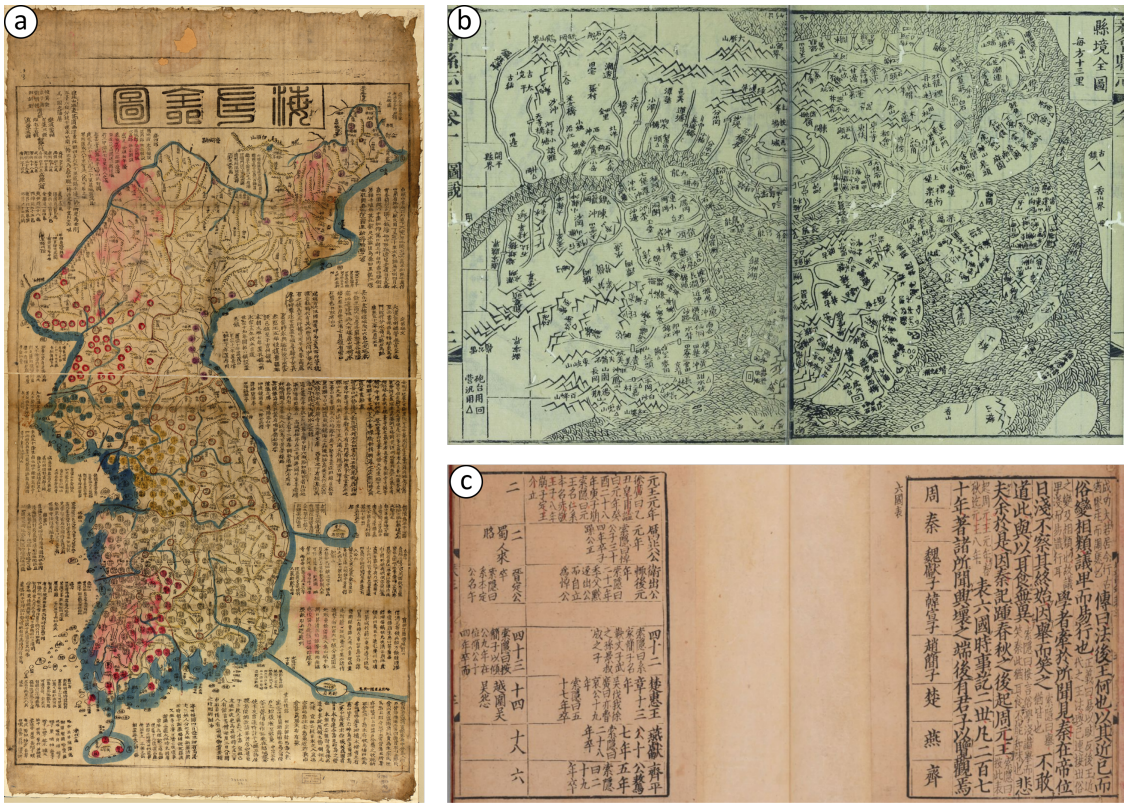


Figure 3.17: Historical charts used by participants: (a) *"Haejwa chŏndo"* published in 1822 showing eight provinces of Korea at the time [264]. The red lines show the admin boundaries of the provinces. The colored dots represent the counties. The counties in the same province share the same color. The black lines connecting counties represent the traffic routes. (b) A 1841 map of Xinhui County recorded in the county's local chronicles [173]. The map was created by Xingzhang Lin, the magistrate of Xinhui County, and Peifang Huang. The landmarks and villages are noted with text. The triangular marks show the distribution of military camps. The double rectangle marks show the distribution of forts. (c) Qian Sima's *"Yearly Chronicle of the Six States"* showing major historical events in the Warring States period of China. The timeline was first published in the book *"Shiji"* around 91 BCE [248]. The timeline reads from right to left. Each column corresponds to a year between 476 BCE and 207 BCE. Each row corresponds to a state. The figure comes from an edition by Shanfu Huang published in 1198 and shows the first two pages of the timeline.

3.7.2 Results

Experience with Historical Visualizations

We observe that historical visualizations, especially maps, are a common source of information for historians. P1 and P3 have used historical maps in their previous research. P1 has used historical maps of Korea when studying the Imjin War, including Fig. 3.17(a). P3 has used maps in local chronicles, including Fig. 3.17(b). Additionally, P3 has used historical timelines, including Fig. 3.17(c). While P4 has not used historical maps in research, P4 has seen many historical maps when reviewing historical documents and developed the hobby of collecting historical maps.

Usage Scenarios of Data Extraction

P1, P3, and P4 commented that our data extraction system could be helpful to their research. They have introduced the following usage scenarios of data extraction. Some scenarios are based on their previous research experience, and others are potential future usage scenarios.

1. **Study spatial correlation of landmarks**: P3 has studied the correlation between distance to mountains and the tendency for villages to join militia groups or peasant revolt in the late Ming Dynasty. It required extracting the location of villages and mountains from maps in local chronicles. Take Fig. 3.17(b) as an example. Extracting villages requires recognizing text labels, and extracting mountains requires recognizing the texture representing mountains. Similarly, P3 and P4 propose scenarios for validating spatial correlations of other landmarks, such as templates, marketplaces, military camps, main roads, and government offices. The extraction of these landmarks mainly requires identifying geometric elements, such as triangles in Fig. 3.17(b) representing military camps.

2. **Study distortions in visualizations**: Distortion and inaccuracy are common in historical visualizations. P4 commented that there are two purposes for historians to use historical visualizations. *" Firstly, historians may want to get the data in these charts and use it as objective data to support their arguments. Secondly, historians may not think that these materials reflect the truth, but they are curious about the imagination of the authors and the reader of the historical visualizations. "* P4 suggested that the distortions of data in historical visualizations also tell stories and are worth studying. They reflect the historical visualization authors' values and may be rooted in the political, economic, and cultural background of the times. For example, P4 hypothesized that *"the areas important to politics, economy and culture are enlarged"*. To verify such an assumption requires extracting regions from historical maps.

3. **Date the creation time of a map**: Historical geography scholars frequently need to date the creation time of historical maps. For some maps, the time may be written. But in many cases, scholars need to infer the time from the map content. P1 suggested that various map features, such as the author information, seals on the map, the toponyms, the naming taboo, the water system, and the admin boundaries, can be used to infer the time ranges of creation. It requires techniques to extract these map features.

In summary, these usage scenarios mainly concern historical maps and involve extracting various map features. Our approach may potentially apply to these scenarios. The cases we demonstrated in Chapter 3 and Chapter 4 also concern extracting data from visual objects embedded in maps. Our systems readily handle some map features, such as rectangular marks, embedded line charts, and embedded Sankey diagrams.

Our implementation may be slightly extended to handle some additional map features the historians concerned. For example, to recognize the military camps in Fig. 3.17(b), we may reuse the implementation for extracting data from Snow's cholera map in Chapter 3. We may have to change the image processing algorithm in the preprocessing step from proposing candidate rectangular marks to proposing candidate triangular marks.

For some of the map features the historians are concerned about, such as text labels, we have considered them in our analysis of reusable workflows in Section 3.6 but have not implemented them yet. The historians' usage scenarios confirm the need to develop more data extraction workflows for these map features in the future, as discussed in Section 6.2. By implementing more data extraction workflows, we can cover more usage scenarios out of the box.

While P1, P3, and P4 believe our system can be helpful to their research, P2 was reserved. P2 mainly used qualitative approaches in research. P2 commented that eyeball estimations might be sufficient for getting qualitative information from historical maps. P2's comments suggest that quantitative evidence gathered from data extraction may be unnecessarily accurate for some qualitative research scenarios.

Potential Improvements

Through the interviews, the participants provide suggestions on extending the data extraction system's functionalities to tailor it to their needs. While most suggestions are outside the scope of data extraction itself but concern data processing after extraction, the suggestions provide more context on how historians may use the extracted data and what they are interested in.

1. **Integration of multiple data sources**: P1, P2, and P3 suggested that the data extracted typically needs to be integrated with other data sources. P1 suggested

that the data extracted from maps, such as text labels, may be cross-referenced to historical textual documents. P2 and P3 suggested geo-referencing historical maps to contemporary maps to view historical landmarks on contemporary maps. P3 suggested the common need to overlay landmarks extracted from multiple historical maps. While the integration of multiple data sources may be done by importing the extracted data into existing GIS systems, it may be beneficial to provide instructions to historians on how to utilize multiple systems to accomplish data extraction and integration goals.

2. **Control on the extracted data structure**: P3 had processed historical timelines such as Fig. 3.17(c). P3 commented that *"the information we want may differ from the table headers given by the authors"*. For example, the timeline may record historical events with rows to represent princedoms and use columns to represent time, while the historians may be interested in the name and position of individuals that got feoffed in each princedom. Thus, P3 needed to *"use keywords to search for relevant information"* and reorganized it into a new table. P3's experience suggests it beneficial to either provide data structure transformation functions within the data extraction system or refer the historians to other data processing systems for them to post-edit the extracted data.

3. **Processing other historical image records**: In addition to historical visualizations, P1 and P3 suggested extending our data extraction approach to other historical image records. P1 suggested the scenario of OCR from historical handwriting on bamboo slips. P3 suggested analyzing common patterns and themes in murals, stone relieves, and portrait bricks. While these scenarios differ significantly from our focus on data extraction from historical visualization, we believe it is a promising direction to investigate using intelligent interactive systems to assist historians in these scenarios.

3.8 Conclusion

We propose MI3, a generic workflow for interactive classification tasks that reduces the user's interaction effort. MI3 targets interactive classification applications with insufficient data to learn a precise classification model. We introduce two components of MI3 for interaction saving: algorithmic sampling and algorithmic default labeling.

We demonstrate how to instantiate the generic MI3 workflow in practice. Specifically, we demonstrate that the scenario of data extraction from historical spatial visualizations, which seemingly is not a classification problem, can be decomposed and converted into three classification subtasks. The MI3 approach is shown to be suitable for all three subtasks, and the three workflows implemented for performing these subtasks are instantiations of the same generic approach.

Based on the decomposition, we develop a prototype software system for data extraction from spatial data visualizations. Its interface can be regarded as an instantiation of MI3's machine-initiated intelligent interaction for supporting active learning as well as performing classification tasks. To evaluate the usefulness of our instantiation of MI3 in the data extraction application, we use the simulation-based approach and compare the number of user operations needed for different solutions to accomplish the interactive classification tasks in data extraction. The simulation results show that MI3's algorithmic sampling and algorithmic default labeling components manage to reduce the required number of user operations. For specific subtasks of data extraction, an instantiation of the MI3 workflow can save up to half of the user operations compared with a conventional ML workflow.

This chapter focuses on the scenario of data extraction from historical visualizations. Section 3.4 has discussed three workflows in data extraction, including block detection, block grouping, and root detection. Section 3.6 has systematically analyzed reusable workflows in data extraction.

Data extraction tasks are related to data labeling tasks. On the one hand, data extraction associates structured datasets with historical visualization images, which can be regarded as a special case of data labeling. On the other hand, our approach in this chapter decomposes data extraction tasks into multiple classification tasks, which are instances of data labeling tasks. For the data extraction workflow in Fig. 3.3, the modules between the preprocessing and postprocessing essentially form a workflow for collecting classification labels with machine assistance. Following the similarities between data extraction and data labeling, Chapter 4 demonstrates the generalization of the idea of integrating machine assistance in this chapter into a broader scenario of data labeling in machine learning.

There are a few limitations and related directions that we hope to address in the future. This chapter's conceptual and engineering contributions demonstrate a significant step toward a potential software system that can help extract data from various visualization images. While the systems demonstrate the feasibility of the approach, there is a need for additional engineering effort to implement eighteen workflows listed in Section 3.6. Specifically, Section 3.4 and Section 3.6.3 has covered in total eight workflows, leaving another ten workflows to be investigated. Moreover, as Section 3.6.2 discussed, there are multiple ways to implement a workflow. A general-purpose data extraction system may need to integrate a couple of alternative implementations of each workflow to accommodate various scenarios. Once there is sufficient coverage of the workflows, the activities for engineering a general-purpose data extraction software toolkit can start.

There is ample scope to develop more and better solutions for various algorithmic modules of MI3, including ML techniques. For the interactive modules of MI3, we have developed only one user interface as an instantiation, and we would like to explore

more design options. We have also assumed that with the termination criterion, the active learning sessions can terminate at a point that minimizes the overall interaction cost. In practice, a naive termination criterion, such as setting a fixed sample size or failure rate or giving the user full control without adequate guidance, may not achieve this goal. In the following, Section 4.3 systematically analyzes the reusable modules in data labeling workflows. The authoring toolkit introduced in Section 4.4, integrates various modules that can be used as alternatives to specific module implementations in this chapter.

In the simulation-based evaluation, we have used a relatively simple assumption that the interaction cost is proportional to the number of user mouse operations. Chapter 5 introduces a refined simulation-based evaluation approach where we use different operator time costs to model different types of user operations. The simulation has also assumed that the user makes no mistakes in carrying out the operations. In the future, one may model the user mistakes to relax this assumption. For example, a simple user mistake model may assume that for each operation, the user mistakes lead to an additional 20% time cost, in which case all our simulated time costs need to be scaled by a factor of 1.2.

"If I can't picture it, I can't understand it."

— Albert Einstein

4

A Toolkit for Data Extraction and Data Labeling

4.1 Introduction

Data extraction from historical visualizations can be seen as a particular type of data labeling task, where a historical visualization image is the data object to be labeled, and a dataset generating the visualization is the label to be assigned. Initially motivated by this observation, we want to investigate whether our approach for reducing user effort in data extraction is reusable and generalizable to other data labeling scenarios (Section 1.2 **O2**).

Labeled datasets play an essential role in supervised machine learning. Modern neural networks are generally trained with large labeled datasets. Data labeling requires human annotators.

Many data labeling systems have been built to enable annotators to label data objects of various types for various learning tasks (e.g., classification [156], segmentation [15]) in different application domains (e.g., image [67], text [242], video [169]). These systems are generally built for a specific labeling task. Different types of labeling tasks require individually built labeling systems.

The research in this chapter is published as "Yu Zhang, Yun Wang, Haidong Zhang, Bin Zhu, Siming Chen, and Dongmei Zhang. *OneLabeler: A Flexible System for Building Data Labeling Tools*. In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI '22). New Orleans, LA, USA. ACM." [299]. The overall idea was developed by Yu Zhang, Yun Wang, Haidong Zhang, and Bin Zhu through discussions. Yu Zhang conducted the interviews, literature coding, implementation, and writing. Yun Wang, Haidong Zhang, Bin Zhu, Siming Chen, and Dongmei Zhang guided the work. Yun Wang, Haidong Zhang, and Bin Zhu revised the writing. This work was conducted during Yu Zhang's internship at Microsoft Research Asia. The code of this work is open-sourced at `https://github.com/microsoft/OneLabeler`.

Building customized labeling systems for various applications is challenging. Programming a labeling system generally requires cross-disciplinary knowledge of interaction techniques and visual design, algorithmic techniques such as active learning and semi-automatic labeling, and software development skills to implement and compose all these modules into a labeling system. It is time-consuming and costly. Although many labeling systems have been built, most of them are monolithic applications with limited usage scenarios and are not designed to support further editing and customization. Adapting an existing system to fulfill a new labeling task with new requirements is generally difficult.

Meanwhile, data labeling systems for different labeling tasks share commonalities. To alleviate the burden of building labeling systems, we advocate a modular composable design that extracts the commonalities among different labeling systems to enable easy customization and extension.

This chapter presents a conceptual framework with a modular composable design for data labeling systems. We have identified eight types of common modules through coding of modules in existing data labeling systems. These modules cover the active sampling and default labeling modules featured in the generic workflow in Fig. 3.3. The framework consists of common conceptual modules and constraints in composing these modules. In the framework, a data labeling system is modeled as a graph denoting its workflow. In the graph, modules are nodes, and the execution order of the modules is encoded with directed edges. The framework generalizes the generic workflow of Chapter 3 into the broader scenario of data labeling in machine learning.

Based on the framework, we have developed OneLabeler, an authoring toolkit to develop interactive intelligent systems for data labeling. OneLabeler is designed with reusability and flexibility in mind. It enables the composition and configuration of software modules. Developers can build a system by creating a workflow graph using built-in implementations of the conceptual modules. In a created system, interface modules and algorithm modules can be instantiated as human, machine, or mixed computation procedures. The constraints on composing modules are embedded in a static program checker to check the workflow graph and verify the feasibility of the created system. A system built with OneLabeler can be exported as an installer for sharing.

To demonstrate the expressiveness of the proposed framework and the OneLabeler toolkit, we present usage scenarios of ten systems built with OneLabeler. These systems cover various usage scenarios, including a set of typical labeling systems for different data types and labeling tasks, a classification system for a customized webpage data type, a machine-aided multi-task text labeling system, and a mixed-initiative image classification system. In addition to these systems addressing common data labeling usage scenarios, Section 4.5.5 demonstrates reproducing Chapter 3's data extraction system. To examine OneLabeler's usability, we have conducted a user study in which developers are asked to accomplish four tasks in building data labeling

systems with OneLabeler. The study results indicate that OneLabeler is easy to learn and use, enabling developers to build different data labeling systems efficiently.

The main contributions of this chapter include:

- a conceptual framework that illuminates **reusable modules** and composition constraints in data labeling systems,
- an **authoring toolkit** based on the framework to support the easy building of diverse data labeling systems, and
- extensive **usage scenarios** to demonstrate the capability of the toolkit in creating diverse systems, including a data extraction system, as well as a **user study** to validate its usability for developers.

4.2 Design Requirements

Various scenarios require deploying different labeling systems to collect labels, but developing an effective labeling system is generally challenging and costly. We see an opportunity to alleviate developers' burden through modular composable design for reuse, as there are commonalities in labeling systems. We aim to develop a flexible system to enable the easy building of various data labeling systems. To better understand the pain points in developing labeling systems, we interviewed three experts with expertise in developing labeling systems as an informal pilot study.

4.2.1 Pilot Study

The three experts we interviewed have developed labeling systems for a company-owned research institute, two with a machine learning (ML) background (E1, E2) and one with an HCI background (E3). They have 8 to 12 years of programming experience. E1 has developed labeling systems for image object detection and binary classification. E2 has developed text labeling systems for relation detection between tables and documents. E3 has developed an image labeling system for multi-label classification. The study started with a structured interview with questions concerning participants' experience in developing labeling systems. Example questions we asked include "Why did you decide to implement a data labeling software instead of using existing ones?" and "What functionalities do you think are critical for data labeling software?" Then, we conducted a walkthrough demonstration of an early prototype of OneLabeler with rudimentary functionalities.

Through the interview, we observed that no participant ever used public labeling software in their projects due to their application-specific requirements. E1 mentioned that he needed to integrate a customized rule to forbid labeling the same data object twice in an image, which could not be achieved in existing systems. E2 mentioned

that the text datasets for the labeling systems he developed were stored in different types of structured data (e.g., document, table, webpage) and required different types of structured labeling. We conclude that customized labeling systems are needed to address application-specific requirements. We conclude that **customization** is a common need to address application-specific requirements on labeling systems.

Developing a data labeling system is time-consuming. It takes 7 to 30 person-days for an experienced engineer to develop one according to the participants' experience. Regarding the three labeling systems he had developed, E2 commented: *"... the interface modules for displaying data points are similar. They all display the table to be annotated, and the supported interactions for annotating the table are similar."* This comment implies that labeling systems share commonalities, but facilitating reuse requires a careful software design. Like other software development, iteration is needed for labeling systems. E1 commented: *"The customized rules typically need to go through many iterations as the ML project progresses."* Similarly, E3 said, *"I didn't know the label categories at the beginning and was not sure about the labeling task that I needed to carry out. Thus, I need to be able to refine the data labeling interface correspondingly as I gradually get more annotations."* Finishing a functional labeling system is not the end of development, as it frequently needs further editing. We conclude that building labeling systems is time-consuming and needs iterations. Meanwhile, there is an opportunity to save the **development cost**, as labeling systems share commonalities.

While E1 and E2, both from the ML background, regarded the annotation interface design and implementation of annotation interaction as the hardest part, E3, from the HCI background, regarded algorithm modules as the hardest to implement. We conclude that grasping all required techniques for implementing a data labeling system can be difficult for labeling system developers since it may involve cross-disciplinary knowledge. Visual programming can help reduce the **skill barrier** on labeling system development in this case [143].

All three experts said they would be happy to use OneLabeler if their labeling tasks were supported. They worked on images, videos, and text for their label tasks. We conclude that the system should build in various modules to support different label tasks for good **coverage of usage scenarios**.

4.2.2 Design Principles and Requirements

The interview confirms the need for a system to support easily building and modifying labeling systems for different labeling tasks and application-specific requirements. It should lower the skill barrier and development cost, cover diverse usage scenarios, and support customization. To achieve this goal, we settle on the following design principles for the system:

DP1 Modular composable design and reuse at multiple levels. To exploit shared

commonalities among labeling tasks and to facilitate reuse and modification, the system should be based on a modular composable design to enable building labeling systems with composable primitives. Primitives should have low coupling. Reuse low-level implementations as composable primitives, and reuse high-level implementations as editable templates. These reuses can significantly reduce development time and effort.

DP2 Ease building and guide the development process. To reduce the skill barrier, the system should facilitate the easy building of labeling systems through visual configuration and guiding developers towards feasible solutions.

Based on the design principles, we propose the following specific requirements for our OneLabeler system. **R1**, **R2**, and **R3** meet **DP1**, while **R4** and **R5** meet **DP2**.

R1 Unified module APIs. Software modules (i.e., interface modules and algorithm modules) should implement unified APIs. Each API should represent a family of common modules in labeling systems. Unified APIs allow substituting a software module in a labeling system or extending the system with a new module without changing other modules (i.e., separation of concerns). In this way, unified APIs make it easier to extend and customize software modules.

R2 Module composition. The system should support the composition of modules to allow a rich space of labeling systems to be created (i.e., good coverage of usage scenarios) with a small number of primitives. Especially, algorithm and interface modules should be composable, as labeling systems typically feature the synergy between humans and machines.

R3 Reuse modules and systems. The system should build in common software modules to be reused as primitives and example composed systems to be reused as boilerplate to scaffold the building of labeling systems, which can significantly ease building commonly used labeling systems and enable extending and customizing built-in labeling systems to fit application-specific requirements.

R4 Visual programming. The system should support visual programming to enable developers to reuse software modules and compose workflows with no or minimal textual coding. Labeling system development requires cross-disciplinary knowledge. A developer may not be proficient with every part of the required technology stacks. Visual programming can reduce the development barrier in such scenarios. It has become a common practice in enterprise machine learning services (e.g., Microsoft Azure[1] and Amazon Web Services[2]). Moreover, the

[1] https://azure.microsoft.com/services/machine-learning
[2] https://aws.amazon.com/sagemaker/

diagram in visual programming provides an overview of the developed system that assists the development process.

R5 Static checking. The system should support static checking to detect infeasible configurations and recommend their fixes to guide editing actions during workflow creation. It enables a developer to spot and localize bugs at an early stage without running the program and guides the developer to reach a feasible solution quickly.

4.3 Analyzing Reusable Modules for Data Labeling

To facilitate fast and easy development of labeling systems, our toolkit should build on a modular design. To design APIs with expressiveness and a suitable level of abstraction, we consult the decomposition of data labeling systems in the literature. In the following, we describe the common conceptual modules and states identified through coding of the literature. Each module, together with its input and output states, defines a unified API for a family of techniques (**R1**). For each module, we give examples of its instances that can serve as software modules in labeling systems (**R3**). An instance of the module corresponds to an implementation satisfying the API. The constraints in composing the modules in building labeling systems are described at the end of the section. The constraints guide the module composition (**R2**) and facilitate static checking (**R5**). Our conceptual framework provides a vocabulary to specify workflows of various existing data labeling systems.

Note that Chapter 3 converts data extraction tasks to data labeling tasks with additional preprocessing and postprocessing. Following this observation, the common modules of data labeling systems can also serve as building blocks of data extraction systems.

4.3.1 Methodology

To identify common conceptual modules, we use data labeling flowcharts in the literature as the dataset and summarize common themes of modules and states of labeling systems. The rationale is that flowcharts depict the cautious decomposition of software into executable modules by the authors. Commonalities in the decompositions likely imply modules with high cohesion and low coupling.

We start with 76 papers on labeling systems and methods published between 2003 and 2021, collected from venues in human-computer interaction, visualization, multimedia, and machine learning. We specifically focus on intelligent labeling systems that use algorithmic approaches to assist the annotator in data labeling.

The full list of the papers is provided in the supplementary material[3]. We aim to summarize a sufficiently presentative set of common modules to be implemented in our toolkit but not to give an exhaustive systematic review. The papers are collected based on our previous experience with data labeling system literature. The top three frequent sources of the papers are ACM CHI (17 papers), ACM MM (8 papers), and IEEE TVCG (5 papers).

For each paper, we extract flowchart figures describing data labeling workflows, which can be application-specific or generic. We discard the papers without flowchart figures and obtain a dataset of 36 flowchart figures extracted from 33 papers (2 of them contain multiple flowchart figures). In the coding process, the workflow figure (Fig. 3.3) we created for the work [297] described in Chapter 3 is one of the 36 flowchart figures.

We aim to derive a software architecture for data labeling systems. Specifically, we want to identify a small collection of categories, with each category corresponding to a set of functionalities that serve similar purposes in different data labeling systems. For example, a sorting algorithm and an active learning algorithm, though named differently, both serve the purpose of ranking the data objects. To this aim, we go through an inductive coding process.

We have conducted the coding of the extracted 36 flowcharts with two stages. In the first stage, we extract the tagged phrases from each flowchart and categorize them into "module" (human, machine, and mixed computation process) and "state" (input and output). The phrases form preliminary sets of module codes and state codes for that flowchart. For example, Crayons [84] supports image pixel-level segmentation with a flowchart containing five phrases. Four of them ("train", "classify", "feedback to designer", and "manual correction") are categorized as preliminary codes for modules because they describe actions and are placed inside blocks in the flowchart figure, while one of them ("interactive use") is excluded as it is a modifier of another phrase.

In the second stage, we group the preliminary codes collected from all the flowcharts into themes. The grouping is not mutually exclusive. We further remove the themes outside the scope of data labeling (e.g., "model understanding" is removed) and merge themes when necessary to synthesize a final code for each theme. The final module/state codes are generated from representative preliminary codes and normalized into short noun phrases describing an action/variable. For example, "classify" in Crayons [84] is finally grouped into the theme "default labeling", and this theme also serves as its final code.

Our coding approach is designed to be inductive to uncover the unknown commonalities [37]. We do not assume a set of codes before the coding process. One of the authors is responsible for coding. Because the author responsible for coding is one of the authors of a paper (i.e., the paper [297] corresponding to Chapter 3) included in the dataset, the coding process inevitably infuses the author's subjectivity. We do

[3]Available at `https://doi.org/10.1145/3491102.3517612`

not see subjectivity as a problem, as our goal is to derive a software architecture, and there is no universal ground truth for software architecture.

We focus on latent codes [37]. We infer underlying semantics and do not purely rely on the explicit meaning of a module from its name given by its author. For example, for modules with the same name and refer to the same implementation in different labeling systems, we still inspect their role in the systems during the coding process, and the modules may be assigned different codes. Looking for latent meanings serves our purpose of identifying the common roles rather than common implementation details. The former is more valuable in informing the software architecture design.

4.3.2 Process

In the coding process, we first extract phrases from the flowchart figures. Each phrase forms a preliminary code. The preliminary codes are first categorized into states and modules. Then, we categorize preliminary codes into themes. Finally, we filter out irrelevant themes and merge related themes into the final codes. In the following, we introduce details of the coding process, including the themes we excluded, the occurrences of final codes in the 36 flowcharts we have coded, and an example of coding a recent labeling system.

Themes

State themes: The 163 preliminary state codes are grouped into 10 themes. 5 of the themes are directly included in the final codes as described in Section 4.3, including **Data Objects**, **Labels**, **Samples**, **Model**, and **Features**. The other 5 themes we identified are (marked with frequency):

- **Labeled data**: the dataset together with (partial) labels (22/163).
- **Search related**: the states related to searching (7/163).
- **Knowledge**: the states related to annotators' knowledge (4/163).
- **Data source**: the source where data objects are extracted (2/163).
- **Others**: the other states not fitting previous themes (9/163).

The theme "labeled data" maps to two final codes "data objects" and "labels" as "labeled data" covers both aspects. There are 2 cases where we group one preliminary code into two themes, where the preliminary code contains the word "with" (e.g., "selected subsets with default labels" in Zhang et al. [297]'s Figure 3).
Modules themes: The 188 preliminary module codes are grouped into 16 themes. 7 of the themes are directly included in the final codes as described in Section 4.3, including **data object selection**, **model training**, **feature extraction**, **default labeling**,

quality assurance, **stoppage analysis**, and **label ideation**. The other 9 themes we identified are (marked with frequency):

- **User labeling**: the process where annotators create/edit data labels (33/188).
- **Labeling interface**: the process to present data (possibly with labels) to annotators for creating/editing labels (17/188).
- **Preprocessing**: the application-specific process to precompute auxiliary data structures used in the data labeling system (11/188).
- **Model understanding**: the process to understand the prediction criteria or quality of the model (5/188).
- **Data exploration**: the process to browse the dataset for familiarizing with the dataset (4/188).
- **Postprocessing**: the process to postprocess data labeling results (i.e., labels) for application-specific needs (4/188).
- **Data collection**: the process to collect data or enlarge the dataset (4/188).
- **User modeling**: the process to model user and interpret user intent (2/188).
- **Others**: the other processes not fitting previous themes (13/188).

The themes "user labeling" and "labeling interface" are merged into the final code "interactive labeling" as the former theme is related to user interaction in interactive labeling, and the latter is related to the interface supporting interactive labeling. For module coding, there are 8 edge cases where we group one preliminary code into multiple themes. It happens typically when the phrase contains the word "and", such as "sampling and annotation" in Liao et al. [169]'s Figure 1. For the preliminary codes for both states and modules, we categorize them into "others" when the preliminary code is hard to categorize, typically when the code is too verbose (e.g., "adding to labeled dataset" in Zhang et al. [297]'s Figure 3) and when the code is too application-specific (e.g., "superpixels" in Zhang et al. [292]'s Figure 2).

State and Module Occurrences

Table 4.1 summarizes the occurrences of final codes for states in the 36 flowchart figures. In the table, for each flowchart figure, if the flowchart contains at least one phrase that corresponds to a specific final code, we add the final code to the "Final Codes" column. When a flowchart figure contains no phrases that refer to inputs/outputs, or the phrases that refer to inputs/outputs are excluded when generating final codes from the themes, we mark the figure's "Final Codes" empty, represented

Table 4.1: Occurrences of states in the literature. The final codes are abbreviated by initials (i.e., "DO" for "data objects", "L" for "labels", "S" for "samples", "M" for "model", and "F" for "features"). Slash (/) is used in final codes when the flowchart contains no states or the flowchart's states are excluded in the coding procedure.

ID	Paper	Venue	Year	Figure Index	Final Codes
1	Fails2003Interactive [84]	IUI	2003	2	/
2	Hoi2005Semi [122]	CVPR	2005	1	DO, L, S
3	Tian2007Face [257]	CVPR	2007	1	DO, L, S, F
4	Cui2007EasyAlbum [67]	CHI	2007	13	DO, L, F
5	Hua2008Online [126]	MM	2008	2	DO
6	Rooij2010MediaTable [230]	CG&A	2010	3	DO, L, S
7	Wang2011Active [270]	TIST	2011	3	DO, L, M
8	Wang2011Active [270]	TIST	2011	5	DO, L, M
9	Wang2011Active [270]	TIST	2011	7	DO
10	Hoeferlin2012Inter [121]	VAST	2012	1(b)	DO, L, M
11	Tang2013Towards [255]	TOMM	2013	2	DO, L, S
12	Zahalka2014Towards [289]	VAST	2014	9	DO, L, M, F
13	Bryan2014ISSE [42]	CHI	2014	5	DO
14	Paiva2015Approach [204]	TVCG	2015	1	DO
15	Russakovsky2015Best [232]	CVPR	2015	2	L, S
16	Liao2016Visualization [169]	TMM	2016	1	DO, L, F
17	Ye2016Face [287]	MM	2016	1	DO, L, S
18	Kucher2017Active [156]	TIIS	2017	1	DO, L, M
19	Ratner2017Snorkel [223]	VLDB	2017	2	DO, L, M
20	Bernard2018VIAL [26]	TVC	2018	1	DO, L, S
21	Bernard2018VIAL [26]	TVC	2018	2	DO, L, S
22	Felix2018Exploratory [89]	UIST	2018	1	L
23	Zhang2018Collaborative [292]	MM	2018	2	DO, L
24	Shang2019Annotating [244]	ICMR	2019	2	/
25	Xiang2019Interactive [280]	VAST	2019	2	DO, L, S
26	Liu2019Interactive [176]	TVCG	2019	3	DO, L
27	Choi2019AILA [64]	CHI	2019	6	/
28	Wang2019DeepIGeoS [268]	TPAMI	2019	1	DO, L
29	Halter2019VIAN [112]	CGF	2019	2	DO, L, S, F
30	Evensen2020Ruler [82]	EMNLP	2020	2	DO, S, M
31	Baeuerle2020Classifier [22]	CGF	2020	1	DO, L
32	Lekschas2020Peax [164]	CGF	2020	3	DO, L
33	Oelen2021Crowdsourcing [201]	IUI	2021	2	DO, L, M
34	Rietz2021Cody [227]	CHI	2021	4	DO, L, M
35	Zhang2021ChartNavigator [293]	TKDE	2021	1	DO, F
36	Zhang2021MI3 [297]	TIIS	2021	3	DO, L, S, M

by a slash (/). Similarly, Table 4.2 summarizes the occurrences of final codes for modules in the 36 flowchart figures.

A Coding Example

AILA [64] supports labeling document classification. Figure 6 in its paper is a flowchart. The figure contains nested blocks, and we use a hyphen to denote the nesting relation in the preliminary code. For example, "preprocessing - stemming" refers to a block named "stemming" placed in a block named "preprocessing". We decompose the nested blocks to the lowest level instead of treating the root block, such as "preprocessing", as a single phrase because different nested blocks may correspond to different themes. The flowchart contains 11 phases, all categorized as preliminary codes for modules because they describe actions. The 11 phrases are grouped into themes and final codes as shown

Table 4.2: Occurrences of modules in the literature. The final codes are abbreviated by initials (i.e., "IL" for "interactive labeling", "DOS" for "data object selection", "MT" for "model training", "FE" for "feature extraction", "DL" for "default labeling", "QA" for "quality assurance", "SA" for "stoppage analysis", and "LI" for "label ideation").

ID	Paper	Venue	Year	Figure Index	Final Codes
1	Fails2003Interactive [84]	IUI	2003	2	IL, MT, DL
2	Hoi2005Semi [122]	CVPR	2005	1	IL, DOS, MT
3	Tian2007Face [257]	CVPR	2007	1	IL, DOS, FE
4	Cui2007EasyAlbum [67]	CHI	2007	13	IL, DOS, FE
5	Hua2008Online [126]	MM	2008	2	DOS
6	Rooij2010MediaTable [230]	CG&A	2010	3	IL, DOS, LI
7	Wang2011Active [270]	TIST	2011	3	IL, DOS, MT
8	Wang2011Active [270]	TIST	2011	5	IL, DOS, MT
9	Wang2011Active [270]	TIST	2011	7	IL, DOS, MT, QA
10	Hoeferlin2012Inter [121]	VAST	2012	1(b)	IL, DOS, MT
11	Tang2013Towards [255]	TOMM	2013	2	IL, DOS, SA
12	Zahalka2014Towards [289]	VAST	2014	9	IL, MT, FE, LI
13	Bryan2014ISSE [42]	CHI	2014	5	IL, MT
14	Paiva2015Approach [204]	TVCG	2015	1	IL, DOS, MT, FE, DL
15	Russakovsky2015Best [232]	CVPR	2015	2	IL, DOS, DL
16	Liao2016Visualization [169]	TMM	2016	1	IL, DOS, MT
17	Ye2016Face [287]	MM	2016	1	MT
18	Kucher2017Active [156]	TIIS	2017	1	IL
19	Ratner2017Snorkel [223]	VLDB	2017	2	MT
20	Bernard2018VIAL [26]	TVC	2018	1	IL, DOS, FE, DL, SA
21	Bernard2018VIAL [26]	TVC	2018	2	IL, DOS, MT, FE, DL
22	Felix2018Exploratory [89]	UIST	2018	1	IL, QA, LI
23	Zhang2018Collaborative [292]	MM	2018	2	IL, DL
24	Shang2019Annotating [244]	ICMR	2019	2	IL
25	Xiang2019Interactive [280]	VAST	2019	2	IL, DOS, DL
26	Liu2019Interactive [176]	TVCG	2019	3	IL, DOS, FE, DL, QA
27	Choi2019AILA [64]	CHI	2019	6	IL, DOS, FE
28	Wang2019DeepIGeoS [268]	TPAMI	2019	1	IL, DL, SA
29	Halter2019VIAN [112]	CGF	2019	2	IL, FE, DL
30	Evensen2020Ruler [82]	EMNLP	2020	2	IL, DOS, MT
31	Baeuerle2020Classifier [22]	CGF	2020	1	MT, DL, QA
32	Lekschas2020Peax [164]	CGF	2020	3	IL, DOS
33	Oelen2021Crowdsourcing [201]	IUI	2021	2	IL, DOS, DL
34	Rietz2021Cody [227]	CHI	2021	4	IL, MT, DL
35	Zhang2021ChartNavigator [293]	TKDE	2021	1	DOS, FE
36	Zhang2021MI3 [297]	TIIS	2021	3	IL, DOS, MT, FE, DL, QA, SA

Table 4.3: Coding the flowchart in AILA [64]. The final codes are abbreviated by initials. A slash (/) in the final code refers to the case that the theme is excluded from the final code.

Preliminary Code	Theme	Final Code
preprocessing - stemming	feature extraction	FE
preprocessing - bag of words	feature extraction	FE
preprocessing - term-document matrix	feature extraction	FE
preprocessing - word vector	feature extraction	FE
preprocessing - sentence vector	feature extraction	FE
document analysis - re-ordering - selecting	data object selection	DOS
document analysis - re-ordering - sorting	data object selection	DOS
document classifier - interactive attentive module - attention weight	preprocessing	/
document classifier - interactive attentive module - prediction score	preprocessing	/
labeling interface - document embedding	data object selection	DOS
labeling interface - document visualization	labeling interface	IL

in Table 4.3. We categorize "attention weight" and "prediction score" as preprocessing because they prepare the data visualized in the labeling interface. "Labeling interface -

document embedding" is categorized as "data object selection" because it refers to an interactive projection where the annotator can select data objects to label.

4.3.3 Results

After the first stage of the coding process, we obtain 188 and 163 preliminary codes for module and state, respectively. For each coded flowchart figure, the number of preliminary module codes ranges between 1 and 11 ($mean = 5.22$, $SD = 2.25$), and the number of preliminary state codes ranges between 0 and 10 ($mean = 4.53$, $SD = 2.61$). After the second stage, we identify five final codes for states (sorted by frequency as marked):

- **Data Objects**: the list of entities to be labeled (59/163).
- **Labels**: the list of annotations assigned to entities (57/163).
- **Samples**: an entity subset annotators handle at a time (18/163).
- **Model**: one or multiple machine-learned models (18/163).
- **Features**: the list of feature representations of entities (13/163).

Similarly, we identify eight final codes for modules: **interactive labeling**, **data object selection**, **model training**, **feature extraction**, **default labeling**, **quality assurance**, **stoppage analysis**, and **label ideation**, which are to be introduced in details in the next subsection. The occurrences of the final codes in the 36 flowcharts are summarized in Section 4.3.2 (Table 4.1 and Table 4.2), and the detailed coding results are included in supplementary material[4].

During the coding process, we have excluded several themes of states and modules regarded as irrelevant to data labeling (see Section 4.3.2). For example, understanding the machine-learned model does not directly benefit labeling, and thus we exclude the "model understanding" theme, whose occurrences include the phrase "understand" in Höferlin et al. [121]'s flowchart. Similarly, the "data collection" theme refers to the process of collecting or enlarging the dataset to be labeled. Its occurrences include the phrase "video crawler" in Hua and Qi [126]'s flowchart. We exclude it because of the low frequency (4 occurrences) and the ambiguity of whether it should be regarded as a part of data labeling or a preparation step before data labeling.

Combining the modules and states finally devised from the coding process, we further identified the common APIs in data labeling systems, as depicted by Fig. 4.1. To ensure that every API has an output, we add two additional states "categories" (output of "label ideation") and "stop" (output of "stoppage analysis"). API inputs are optional, as a software module implementing an API may utilize some or even

[4]Available at `https://doi.org/10.1145/3491102.3517612`

none of the inputs available. These API definitions are integrated into OneLabeler (described in Section 4.4).

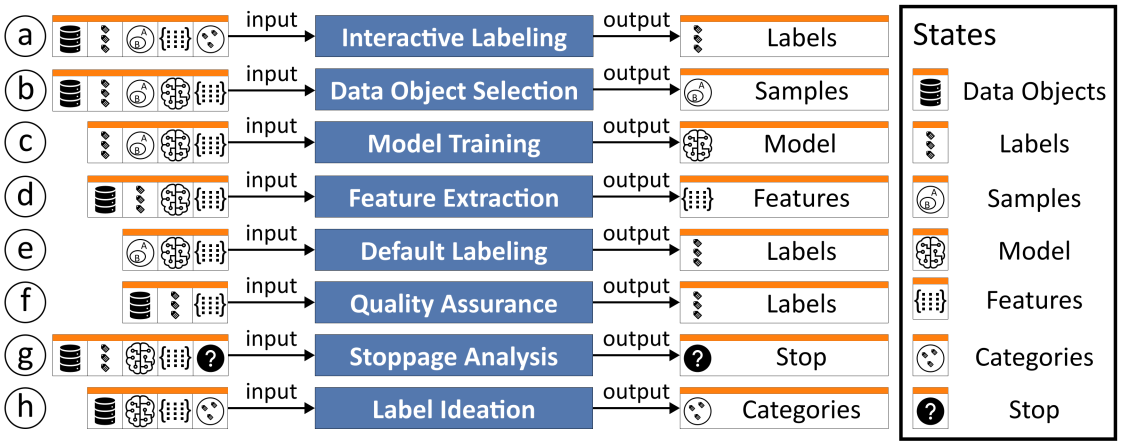


Figure 4.1: Common APIs in data labeling systems.

4.3.4 Common Modules

The conceptual modules (sorted by frequency as marked) and their instances as software modules are described below. An instance can be an algorithm (i.e., a machine computation process), an interface module involving human computation, or a mixed computation process.

- **Interactive Labeling** (49/188) is the core module where annotators carry out labeling tasks in an interface (Fig. 4.1(a)). Different instances of this module typically vary in the design of task, interface, and interaction. *Task design* concerns the task that annotators are instructed to carry out. Typical tasks include label assignment and correction tasks for the concerned data type (e.g., image, text) and label task type (e.g., classification, segmentation). Alternatives include GWAP instructing annotators to play a game [4–6]. When strong labels (e.g., segmentation) are needed, using weakly supervised learning may allow annotators to provide weak labels (e.g., classification) that are then algorithmically compiled to strong labels, potentially improving labeling efficiency [36]. Additionally, much research effort in crowdsourcing concern task scheme design [52]. *Interface design* typically concerns data objects' layout, such as grid layout and hierarchical layout [67, 290]. Tasks requiring fine-grained editing may prefer displaying one data object each time [15, 233]. *Interaction design* concerns interaction techniques facilitating efficient labeling such as batch edit [253] and graph cut [35].

- **Data Object Selection** (34/188) determines the order for data objects to be selected and labeled by annotators (Fig. 4.1(b)). Instances include active learning

strategies that select informative data objects first [242]. The selection may also use clustering algorithms to group similar data objects [67, 174], which may enable annotators to assign the same label in one go. Annotators may involve in deciding which data objects to label [43, 169].

- **Model Training** (21/188) trains/updates a learning model (that may serve as input to other modules) with newly gathered labels (Fig. 4.1(c)). Variations of model training may concern what model is trained and how the training is conducted. Although any predictive model may serve the purpose, semi-supervised learning and transfer learning techniques better match data labeling scenarios where unlabeled data objects are abundant (useful for semi-supervised learning), and the cold start issue is prominent (alleviated by transfer learning). Model training may be conducted by training from scratch or by incremental update methods in online learning.

- **Feature Extraction** (20/188) turns data objects into feature representations, typically vectors (Fig. 4.1(d)), facilitating other modules that cannot work with raw data objects (e.g., model training). Algorithmic instances of feature extraction can be handcrafted (e.g., HoG), unsupervised (e.g., PCA), or supervised (e.g., LDA). It is also possible to involve annotators in feature extraction [58].

- **Default Labeling** (17/188) assigns tentative labels to data objects, simplifying annotators' work from creating labels to verification and correction (Fig. 4.1(e)). It may be facilitated by models trained with the model training module, pre-trained models, or rules.

- **Quality Assurance** (6/188) reviews label quality and corrects erroneous labels (Fig. 4.1(f)). Algorithmic relabeling may be used to suggest suspicious labels for annotators to verify [40, 200, 306]. Annotators may also exploratorily review labels and search for potentially mislabeled data objects to correct [22, 101, 280]. In quality-sensitive applications, quality assurance may require going through all the labels one by one [298].

- **Stoppage Analysis** (4/188) decides whether to keep assigning tasks to annotators or stop (Fig. 4.1(g)). A common criterion is to check if all the data objects have been labeled once. Alternative criteria may decide the stoppage time by empirical measures of the label quality [72] or confidence [303] and stability of models [29, 294] trained with the partially labeled dataset.

- **Label Ideation** (3/188) develops the label categories used for labeling (Fig. 4.1(h)). It may appear as an interface widget allowing an annotator to create new categories ad hoc. More structured ways to generate categories may involve al-

gorithmic assistance (e.g., topic modeling) in an interface that supports users iteratively propose, verify, and refine categories [89, 158].

Through the coding process, we observe that interactive labeling is by far the most common module in labeling systems. It appears 49 times among the 188 codes extracted from the 36 flowcharts, meaning that on average more than one module in each flowchart is related to interactive labeling. Any flowchart that strictly depicts a data labeling workflow with human annotators would need to mention it at least once.

Note that the theme "label ideation" above is not involved in our workflow for data extraction described in Figure 3.3. In the data extraction workflow, the task concerned is a binary classification task that involves two label categories. As it is known that there are two label categories, label ideation is not needed for the data extraction workflow.

4.3.5 Module Composition Constraints

Labeling systems can be built with instances of the above conceptual modules as building blocks. However, not all compositions of software modules produce a valid labeling system. A labeling system is represented by a flowchart that contains module nodes, i.e., nodes denoting implementation of common modules, as well as initialization, decision, and exit nodes. Assuming that states of a labeling system are stored globally, modules fetch inputs from and register outputs to corresponding global states.

We design the following constraints for composing software modules into a valid labeling system:

- **Valid Flowchart**: The graph should satisfy graph-theoretic constraints that generally hold for a flowchart [140]. For example, all the nodes should be reachable from the initialization node.

- **Input Initialized**: For each possible walk on the graph, a node that represents a software module should not be visited until its input parameters are all initialized.

- **No Redundancy**: After a module is visited, it should not be revisited until at least one of its inputs has changed its value. After a module is visited, its output(s) should be used by a module.

- **Involve Labeling**: Interactive labeling should exist in all walks of the graph to ensure it depicts a workflow of a labeling system.

The first two constraints ("valid flowchart" and "input initialized") are derived from the rationale that the flowchart should represent a correct program. The third constraint concerns the efficiency of the software. Labeling systems often require

heavy human computation (e.g., manual annotation) and heavy machine computation (e.g., model training), making performance optimization a critical issue. The fourth constraint ensures that the represented software is a labeling system. These constraints translate to rigid propositions (in Appendix A) and are integrated into OneLabeler's static checking function to validate user-created labeling workflows. The list of modules and module composition constraints essentially form a design space of data labeling systems.

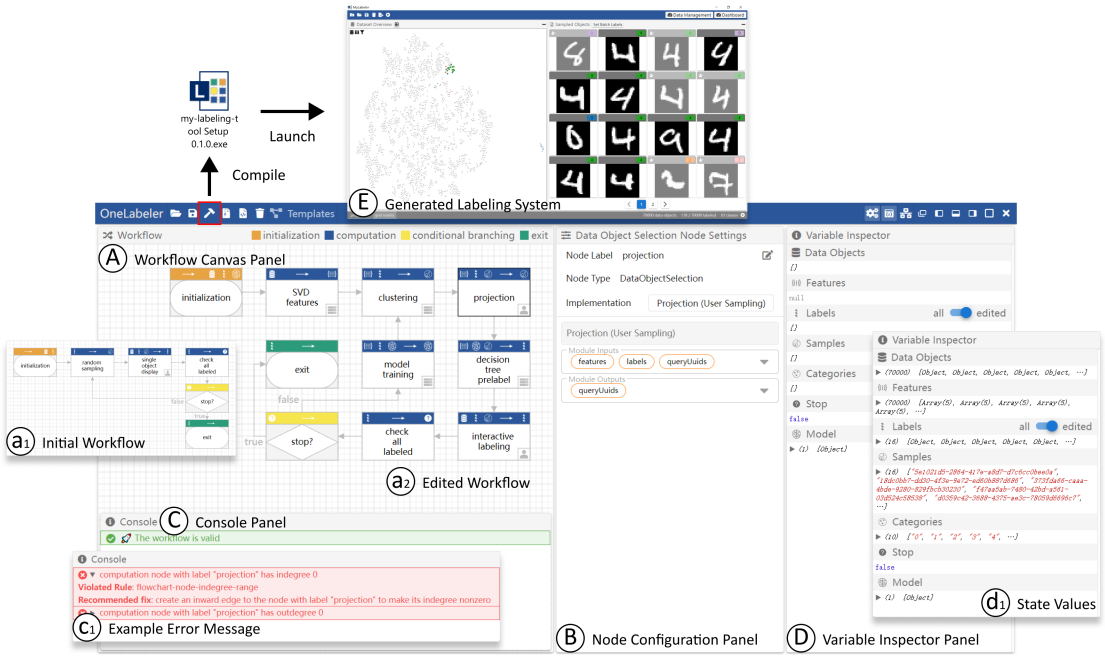


Figure 4.2: OneLabeler's visual programming interface: (A) Workflow canvas panel for a developer to create and compose nodes, as well as editing the visual layout of the workflow; (B) Node configuration panel for configuring parameters of a selected node; (C) Console panel for showing static checking result of whether the workflow denotes a valid labeling system; (D) The variable inspector panel showing state values; (E) The generated data labeling system by OneLabeler according to the workflow.

4.4 Authoring Toolkit

Based on the requirements described in Section 4.2, we propose the OneLabeler system for building labeling systems. OneLabeler enables developers to visually program (**R4**) data labeling systems by composing software modules into a workflow (**R2**). A created labeling system can be exported as an installer. The conceptual modules identified in Section 4.3 inform the API design for data labeling modules in OneLabeler (**R1**). A developer can reuse a collection of built-in modules and templates (**R3**) or customize on demand. OneLabeler's visual programming environment supports static checking (**R5**) of user-created workflows to assist debugging and guide towards feasible solutions.

4.4.1 System Architecture

Module

The eight types of conceptual modules (interactive labeling, data object selection, model training, feature extraction, default labeling, quality assurance, stoppage analysis, and label ideation) are integrated into OneLabeler as eight API definitions (**R1**). An API can be implemented with either an algorithm or interface module. While an algorithm module can be automatically executed, the execution of an interface module requires human intervention. In the following, we use the data object selection module as an example for illustration.

Defining a conceptual module as API: As shown in Fig. 4.1(b), the data object selection module conceptually depicts a function that returns a dataset subset, given data objects, labels, features, model, and samples. The inputs are optional. Schematically, it translates to an API definition in OneLabeler as:

```
class DataObjectSelection {
  /** Input states. */
  props: ['dataObjects', 'labels', 'samples', 'model', 'features'];
  /** Output state's setter, evoked in .render or .run. */
  methods: ['setSamples'];
  /** Whether the execution of the module is blocking or not. */
  blocking: boolean;
  /** (Optional) Only needed for an interface module. */
  /** Whether to render the module when not executed. */
  persistent?: boolean;
  /** (Optional) Interface module implementation. */
  render?: Function;
  /** (Optional) Algorithm module implementation. */
  run?: Function;
}
```

Executing a module implementation: Conceptually, executing a module implementation is to send it the input states and wait for the output states to be returned. For example, the data object selection API defined above can be implemented with an algorithm (e.g., an active learning strategy) or a user interface module (e.g., a projection scatterplot as Fig. 4.5(B_1) where an annotator can select manually with lasso). When the API is implemented as an algorithm, the execution is a straightforward function call. When the API is implemented as an interface, the execution is regarded as a function call that has the side effect of altering the user interface. The function returns when a callback function setting the returned states is triggered by user interaction in the interface. For example, a projection scatterplot (as Fig. 4.5(B_1)) is regarded as a function that returns the selection after the user finishes a lasso selection in the scatterplot. Data object selection takes samples as an optional input because when implementing it with a projection scatterplot, the scatterplot may need to highlight the previous samples.

Input and output access: Each module fetches inputs and returns outputs through getters and setters exposed by a global data model. We refer to this storage model

as a *blackboard model* as all the modules read from the blackboard and write on the blackboard. OneLabeler and labeling systems generated with OneLabeler follow the model-view-controller (MVC) design pattern[5]. The seven states of labeling systems in Fig. 4.1 are stored globally and managed by the data model. The data fetching and storage are configurable in labeling systems built with OneLabeler. It allows multiple annotators to use installations of a labeling system that have data access from the same database, enabling simultaneous annotation by multiple annotators.

Workflow

To enable developers to create complex systems with a small number of primitives, OneLabeler supports the composition of software modules into a workflow declaring a labeling system (**R2**).

Defining a workflow: Within OneLabeler, a labeling system is declared by its workflow graph with modules being nodes and execution order specified by directed edges. The graph can be either created in OneLabeler through visual programming (see Section 4.4.2), or uploaded to OneLabeler through a JSON file storing a list of nodes and edges. A valid workflow graph has to satisfy constraints described in Section 4.3.5. Unlike data flow systems [185, 288], directed edges in OneLabeler only define execution order without implying data transmission. Each module fetches inputs and returns outputs through the global data model. Thus, for each directed edge, the source node's output is not necessarily the target node's input.

Executing a workflow: To execute a workflow, OneLabeler traverses the graph representing the workflow. The traversal starts from the initialization node in the graph. OneLabeler recursively visits subsequent nodes following directed edges. When a node corresponding to a module is visited, OneLabeler supplies the module's required inputs by reading the blackboard model and executes the module as a function. When the module is an interface module, the interface is rendered in a window (e.g., Fig. 4.5(B_1) is a rendered window for data object selection using interactive projection). By default, OneLabeler waits for the function to return. After the function returns, OneLabeler registers the return value to the global data model and then visits the next node. Alternatively, the module can be optionally configured to be non-blocking, in which case OneLabeler visits the next node before the function returns. When the workflow is faithfully executed, the labeling system's interface may constantly change because, after execution, an interface module will no longer show up until the next visit. The "persistent" option addresses this issue of the rapid context switch. For a "persistent" interface module, the interface is persistently rendered in the window even when the node is not executed. When a decision node is visited, the decision criterion is

[5]To disambiguate, the (data) "model" here refers to the part of software responsible for managing states (e.g., states in Fig. 4.1), instead of a machine learning "model".

checked, and the edge to follow is chosen accordingly. The execution of the labeling workflow terminates when the exit node is visited.

4.4.2 Visual Programming of Workflows

OneLabeler enables developers to build labeling systems through visual programming of the labeling systems' workflows (**R4**). The navigation toolbar provides import/export and compilation functions through button clicking (highlighted in Fig. 4.2 with red). The labeling system's workflow can be imported/exported in the JSON format to facilitate reuse and sharing. A developer can use the built-in templates provided in the template menu as a boilerplate to scaffold development. When the workflow is finalized, the developer can click the compile button on the navigation bar to compile the workflow into an installer of the corresponding labeling system.

Visual Programming and Configuration

In OneLabeler, a data labeling system is declared by its workflow. To enable developers to visually configure the workflow without textual programming (**R4**), OneLabeler provides a visual programming interface (Fig. 4.2). Within the interface, the developer can build a labeling system by interactively configuring its workflow.

Adding a node: Within the workflow canvas panel (Fig. 4.2(A)), the developer can add nodes through the right-click menu on the canvas. The menu provides module nodes and control nodes. The module nodes belong to one of the eight types of conceptual modules (in Section 4.3). The control nodes can be initialization, decision, and exit nodes. After creating a module node, to make it executable, the developer needs to configure the implementation used for the node, as described below.

Composing nodes: Two nodes can be composed by creating a directed edge to specify the execution order. To create an edge between two nodes, the developer can hover on a node serving as the source, which shows the link ports of the node, drag the link port of the node, and release on the link port of another node serving as the target.

Configuring a node: To configure the implementation details of a node, the developer first needs to select a node in the workflow canvas panel by clicking. The node configuration panel (Fig. 4.2(B)) shows the details for the selected node. The user can select the built-in implementation of the node in the "implementation" menu in the configuration panel. The parameters of the selected implementation can be configured on demand. For example, for the data object selection module (as shown in Fig. 4.2(B)), the user can configure which selection method to use (e.g., active learning or interactive projection) and the parameters of the selected implementation (e.g., how many data objects to sample each time by the active learning strategy). Additionally, the outputs of the initialization node are configurable. If a state is configured as the initialization node's output, it will be initialized to a non-empty value when initialization is executed.

Static Checking

To guide a developer smoothly towards a feasible solution (i.e., a workflow that depicts a valid labeling system), we use the classic idea of static program analysis [17] in software debugging (**R5**). The constraints (in Section 4.3.5) allow OneLabeler to check the graph-theoretic properties of the workflow created by the developer as a proxy of the feasibility of the created labeling system. Besides, we add practical constraints on the graph data structure (e.g., node id should be unique) and on module configuration (i.e., an implementation should be chosen for the module). OneLabeler integrates these constraints into a checker to statically validate the created labeling system. OneLabeler notifies the developer of identified violations of the constraints in the console panel (Fig. 4.2(C)), allowing the developer to identify potential mistakes before using the created labeling system. Hovering/Clicking an error message in the console panel highlights/selects the node(s) and edge(s) involved in the error in the workflow canvas panel. Clicking the triangular expand button on the error message shows the error code and the recommended way(s) of fixing the error (Fig. 4.2(c_1)). For example, when a new module node is just created by the developer, there is no edge connecting it to other nodes. In this case, the "valid flowchart" rule (in Section 4.3.5) is violated, since this node has indegree and outdegree zero, meaning it is not used. The error messages *computation node with label "projection" has indegree 0* and *computation node with label "projection" has outdegree 0* displayed in the console panel remind the developer to resolve this issue. OneLabeler continuously validates the workflow as the developer edits it. The messages are ranked by severity. The low-severity errors are hidden from the console panel until high-severity ones are resolved to avoid overwhelming the developer with many error messages. The developer can arrive at a feasible solution, i.e., a labeling system that works, by iteratively resolving the error messages.

Labeling System Preview

To help the developer debug the built labeling system, OneLabeler provides a real-time preview of the built system. As the developer finishes the workflow, the developer can try out the preview to examine if it meets the requirement. During the process, the developer can inspect the state values (e.g., Fig. 4.2(d_1)) of the labeling system through the variable inspector (Fig. 4.2(D)). OneLabeler also enables the developer to manipulate the control flow of the preview for debugging. Specifically, the developer can conduct single-step debugging for a node or force the control flow to start from a node. The control flow manipulation functions can be selected in the node's right-click menu.

4.4.3 Built-in Modules

OneLabeler provides a collection of built-in implementations for the conceptual modules. The user can configure the implementations on demand. Moreover, for

interactive labeling modules, the user can configure three dimensions (i.e., data type, label task type, and interface design) to derive various combinations.

- For **interactive labeling**, as it is the most important module in data labeling, OneLabeler splits it into three design dimensions: data type, label task type, and interface design. OneLabeler builds in the following implementations for the three dimensions:

  - **Data type**: image, text, video, audio, point cloud.
  - **Label task type**: single-/multi-label classification, freeform text annotation, object detection (for image), segmentation (for image and point cloud), text/temporal span tagging (for text, audio, and video), span relation (for text, audio, and video).
  - **Interface design**: single object display (e.g., Fig. 4.4(B_1)) and grid-based interface with editable layout (e.g., Fig. 4.5(B_2)).

  The three dimensions are identified following the rationale that the interface for interactive labeling needs to show data object details (which depends on the **data type**), provide the interaction for annotating labels (which depends on the **label task type**), and layout the data objects following an **interface design**. The data types are chosen to cover the data types in the 33 coded papers as well as common data types in machine learning benchmark datasets as indexed by *Papers With Code*[6], including image, text, video, audio, point cloud, and sequential data. We have implemented all these data types except sequential data as it is concerned in only one paper [164]. Similarly, the built-in label task types cover all the tasks in the 33 papers except for video object tracking with one occurrence [121]. The two most common interface designs (i.e., grid-based interface and single object display) in the 33 papers are already built-in, while others (i.e., thumbnail projection, table with metadata, and thread layout) are excluded for now. Those excluded from the current built-in can be implemented in the future.

- For **data object selection**, OneLabeler builds in algorithmic selection and interactive selection implementations. For the algorithmic selection, OneLabeler provides active learning techniques (including three entropy-based methods [38, 167, 281], least confidence, and smallest margin), clustering-based techniques (selection by ranking cluster labels, distance to cluster centroids, and density estimation). For interactive selection, OneLabeler provides interactive configurable data projection methods (e.g., Fig. 4.5(B_1)). The projection can be configured

[6] https://github.com/paperswithcode

to be a scatterplot or heatmap of a T-SNE/PCA/LDA projection of the feature values, where the user can select data objects to label.

- For **model training**, OneLabeler currently builds in several classic supervised/semi-supervised learning algorithms, including decision tree, support vector machine, logistic regression, restricted Boltzmann machine, and graph-based label propagation. OneLabeler can be easily extended to support models provided by libraries that follow the scikit-learn library's API design.

- For **feature extraction**, OneLabeler builds in three techniques for image feature extraction, including a bag of features technique (handcrafted features describing color, edge, and texture), truncated SVD of raw images (unsupervised), and LDA projection of raw images (supervised). For text feature extraction, non-negative matrix factorization of tf-idf is provided.

- For **default labeling**, OneLabeler builds in model prediction, where the model can be the ones trained by the model training module(s) or provided by the developer through a prediction API. OneLabeler also provides a rule-based default labeling method for text span labeling based on part-of-speech (POS) tagging.

- For the other three conceptual modules with relatively low frequencies (in Section 4.3), OneLabeler builds in basic implementations for the time being. For **quality assurance**, modules for interactive labeling can be reused for the annotator to go through data objects one by one. For **stoppage analysis**, OneLabeler builds in a criterion based on the sample rate specifying the rate of data objects to be labeled by annotators before stopping. For **label ideation**, OneLabeler provides an interface widget where label categories can be dynamically added, and for each label, the applicable label task type can be specified.

A developer can craft a labeling system's workflow in the visual programming interface from scratch with the modules described above. Additionally, OneLabeler builds in a collection of predefined template workflows, covering combinations of built-in data types and label task types. Specifically, the built-in templates include image classification, image segmentation, text classification, text span tagging, video classification, video temporal segmentation, audio classification, audio temporal segmentation, point cloud classification, and point cloud segmentation. The built-in templates are chosen to cover the five built-in data types, each data type with one coarse label task (i.e., classification) and one fine-grained label task (e.g., segmentation). A developer can start with a predefined template and tailor it towards specific needs. For example, a developer may start from a template workflow shown in Fig. 4.2(a_1) and add nodes to support default labeling and mixed-initiative sampling

and corresponding connections, resulting in a revised workflow shown in Fig. 4.2(a_2). The workflow (Fig. 4.2(a_2)) and the resulting data labeling system (Fig. 4.2(E)) is almost the same as the ones detailed in Section 4.5.4 except for minor differences in the configuration for interactive projection and cluster sampling batch size.

4.4.4 Customization

OneLabeler's modular architecture supports extensions for module implementations and workflow templates (**R1**). If the built-in implementations do not meet a developer's needs, the developer can provide customized implementations. OneLabeler provides a command-line interface (CLI) to scaffold customization. Using the CLI, the developer can choose a type of customization as introduced below, and OneLabeler will create corresponding template files for the developer to start with (see our code repository at `https://github.com/microsoft/OneLabeler` for details on the CLI).

General-Purpose Module Customization

To add a customized module to OneLabeler, a developer needs to fill the data structure as specified by the API definition in Section 4.4.1. For an algorithm module, a developer needs to implement the "run" function that includes the code for running the algorithm and registering the result through setters (e.g., "setSamples") when the algorithm returns. For an interface module, a developer needs to implement the "render" function that includes the code for rendering the module and registering the result through setters in the callback function of user interaction events. For a customized algorithm module implemented as an algorithm server, OneLabeler allows the module to be directly registered in the visual programming interface.

Customization for Interactive Labeling

OneLabeler internally represents a built-in interactive labeling module with three submodules: data type, label task type, and interface design. Customizing interactive labeling modules can be done by following the instructions above for general-purpose customization or by implementing only a customized submodule. To customize the data type, a developer needs to provide a display function defining how to display a single data object of this data type in the interface design. This display function takes as input the data object's data structure, together with the width and height of the display area for it. For example, to extend OneLabeler to support webpage labeling, a developer may specify that a data object of the webpage data type should be displayed with the HTML iframe tag (see details in Section 4.5.2). To customize the label task type, a developer needs to provide a display function defining the interaction widgets (e.g., buttons and menus). These widgets are appended to the toolbar in the interface design to support the labeling interactions. For the label task type that requires altering

the visual appearance of the data object (e.g., image segmentation requires displaying a segmentation mask), a developer additionally needs to provide a render function defining how the created annotation of a data object should be displayed. To customize the interface design, a developer needs to implement a display function defining the annotation interface. The display function of the annotation interface takes the display function of data type declarations and toolbar and the rendering function of label task type declarations as input.

Template Customization

To extend OneLabeler with a customized workflow template, a developer can either provide a TypeScript/JavaScript file declaring the workflow as an object or an equivalent JSON file. Below is the code snippet to declare a customized workflow template for webpage classification.

```
/** The declaration of a minimal webpage classification workflow. */
export default {
  nodes: [
    {
      label: 'initialization',
      type: 'Initialization',
      value: {
        dataType: 'Webpage',
        labelTasks: ['Classification'],
      },
      layout: { x: 40, y: 40 },
    },
    {
      label: 'random sampling',
      type: 'DataObjectSelection',
      value: ['Random', { params: { nBatch: { value: 2 } } }],
      layout: { x: 160, y: 40 },
    },
    {
      label: 'grid-based interface',
      type: 'InteractiveLabeling',
      value: ['GridBasedInterface', {
        params: {
          nRows: { value: 1 },
          nColumns: { value: 2 },
        }
      }],
      layout: { x: 280, y: 40 },
    },
    {
      label: 'check all labeled',
      type: 'StoppageAnalysis',
      value: 'AllLabeled',
      layout: { x: 400, y: 40 },
    },
    {
```

```
      label: 'stop?',
      type: 'Decision',
      layout: { x: 400, y: 130 },
    },
    {
      label: 'exit',
      type: 'Exit',
      layout: { x: 410, y: 220 },
    },
  ],
  edges: [
    { source: 'initialization', target: 'random sampling' },
    { source: 'random sampling', target: 'grid-based interface' },
    { source: 'grid-based interface', target: 'check all labeled' },
    { source: 'check all labeled', target: 'stop?' },
    { source: 'stop?', target: 'exit', condition: true },
    { source: 'stop?', target: 'random sampling', condition: false },
  ],
}
```

4.4.5 Workflow Compiler

OneLabeler enables a workflow to be compiled into an installer of the specified labeling system through button-clicking in the visual programming interface. The compilation is conducted by the back end of OneLabeler that holds a mirror of OneLabeler's source code. Upon receiving the compilation request, the back end hard-codes the workflow in the request into the source code mirror. Then, the back end filters out from the mirror the dead code irrelevant to the labeling system declared by the workflow (e.g., the built-in modules of OneLabeler not used by the labeling system and the code responsible for OneLabeler's workflow editing/compilation). The modified and filtered version of OneLabeler's source code is then compiled into the declared labeling system's installer using Electron Packager[7]. The installer installs the labeling system as desktop software.

4.5 Usage Scenarios

OneLabeler builds in various implementations of common modules in data labeling systems as introduced in Section 4.4, which can directly lead to a wide range of data labeling systems with appropriate configurations. To demonstrate the effectiveness of OneLabeler in supporting the development of data labeling systems, this section presents ten sample data labeling systems built with OneLabeler.

We first introduce a gallery of six basic labeling systems based on built-in data types and label tasks. Without changing the topology of a simple workflow template in Fig. 4.7(A), the six example systems can be directly generated and can reproduce

[7] https://github.com/electron/electron-packager

major functionalities of popular open-source labeling systems that cover various data types (including image, text, video, audio, and point cloud) and labeling tasks (such as classification, multi-label classification, freeform text annotation, segmentation, span tagging, and span relation).

Moreover, following the API of modules as described in Section 4.3, developers can also extend the capabilities of OneLabeler by providing new implementations as plugins. To further demonstrate the expressiveness and extensibility of OneLabeler, we showcase four more examples of advanced data labeling and interactive machine learning applications that integrate additional algorithm and interface modules, requiring more complicated workflows.

4.5.1 A Gallery of Basic Labeling Systems

In Fig. 4.7, we present six examples of basic labeling systems without machine assistance, based on the same workflow template in Fig. 4.7(A). With corresponding modifications to the configurations of the interactive labeling module, concerning data type, label task type, and interface design, we can achieve specific data labeling systems that target various tasks, such as classification (Fig. 4.7(B)), segmentation (Fig. 4.7(C)), span tagging (Fig. 4.7(D)) in different application domains including image (Fig. 4.7(B) and 4.7C), text (Fig. 4.7(D)), video (Fig. 4.7(E)), audio (Fig. 4.7(F)), and point cloud (Fig. 4.7(G)). In Fig. 4.7, we load public datasets to the created labeling systems, including Imagenette[8] (Fig. 4.7(B) and Fig. 4.7(C)), IMDb reviews [179] (Fig. 4.7(D)), KTH Action [237] (Fig. 4.7(E)), LibriSpeech [205] (Fig. 4.7(F)), and ModelNet [279] (Fig. 4.7(G)).

The example systems can reproduce the capabilities of some existing data-labeling systems in public use. For example, the labeling system in Fig. 4.7(C) supports image classification, multi-label classification, freeform text annotation, object detection (with polygon or bounding box), and segmentation. These functionalities cover the major functionalities of LabelImg [172] (for object detection with bounding box) and LabelMe [233] (for object detection with polygon).

Similarly, using the text labeling system in Fig. 4.7(D), major features of Doccano [192] can be reproduced. Doccano supports labeling of image classification and multiple text-related label tasks, including named entity recognition, sentiment analysis, translation, and text to SQL. With the text labeling system in Fig. 4.7(D), named entity recognition can be done by span tagging, sentiment analysis can be done by classification, while translation and text to SQL can both be done by freeform text labeling.

4.5.2 Customizing a Webpage Data Type

OneLabeler can be extended with customized data types. A customized data type can be implemented as a submodule of the interactive labeling module (Section 4.4.4).

[8] https://www.tensorflow.org/datasets/catalog/imagenette

Customizing a data type requires specifying how a single data object should be displayed. For example, to add a new "webpage" data type, one can supply the following JavaScript code (by putting the code file in the data type folder of OneLabeler system). The API design of data type declaration is compatible with Vue.js[9], a popular front-end development framework, enabling developers to utilize their previous programming experience to develop OneLabeler's plugins.

```
/** The declaration of webpage data type. */
export default {
  /** The label of the data type to appear in the menu. */
  label: 'webpage',
  /** Inputs to the display function. */
  props: ['dataObject', 'width', 'height'],
  /** Render a webpage using HTML iframe element. */
  render: (h, { props }) => (
    h('iframe', {
      attrs: {
        width: props.width,
        height: props.height,
        src: props.dataObject.src,
      },
    })
  ),
}
```



Figure 4.3: A classification system for the customized webpage data type.

With the corresponding data type declaration, OneLabeler can support a new data type with tasks independent of data types, such as classification, multi-label

[9]https://vuejs.org/guide/extras/render-function.html

classification, and freeform-text annotation. For example, using the webpage data type, we can build a basic webpage classification system as shown in Fig 4.3. This system originates from the same workflow as Fig. 4.7(A), except that the data type is set to the newly customized webpage data type.

Developers can further develop data-type-specific label tasks with additional code. For example, span tagging is originally targeted at text, video, and audio data types. By specifying how to select a text "span" in HTML text nodes inside a webpage, the developer can make span tagging applicable to webpages.

4.5.3 Machine-Aided Multi-Task Text Labeling

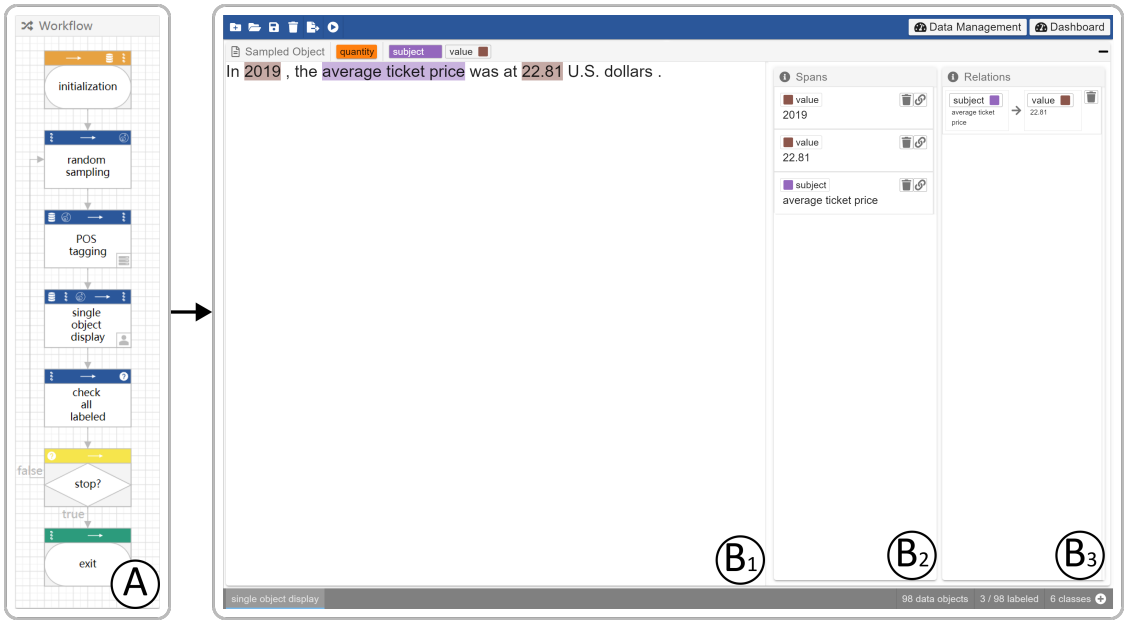


Figure 4.4: Building a machine-aided multi-task text labeling system: (A) User-configured workflow for the system. The interface of generated multi-task text labeling system: (B_1) A labeling panel showing the sentence for the annotator to conduct **interactive labeling**; (B_2) A list of created named entities; (B_3) A list of created entity relations.

Here, we introduce a configured text labeling system that incorporates machine assistance. This system is motivated by a data labeling scenario in Cui et al. [68]'s supervised technique that learns from a human-annotated text dataset. The data objects are quantitative statements. The label tasks require annotating the statement type (i.e., document classification), the subject and value words in each sentence (i.e., span tagging), and the correspondence between subject and value words (i.e., span relation detection). In this usage scenario, the three types of text labels described above are required to be assigned to sentences of quantitative statements. Therefore, we demonstrate the creation of a system that enables annotators to provide three types of labels: document classification, span tagging, and span relation detection.

In this application, the concerned data objects are text (i.e., strings), and the label tasks are to (1) assign one of four class categories ("proportion", "quantity", "change", and "rank") to the text to mark what type of statement it belongs to, (2) extract named entities from text and associate them with one of two class categories ("subject" and "value"), and (3) link pairs of named entities if they are related subject and value.

Semi-automatic labeling [15, 101, 297] is a common technique in the literature for efficient annotation. To incorporate this idea in the labeling system, we can start from the simple labeling workflow template as shown in Fig. 4.7(A), add a default labeling module, and configure it to be implemented with the built-in POS tagger. Specifically, the developer may decide to use a predictive model to detect tentative text spans that belong to the subject and value words. In this way, the annotator may save much effort when the predictive model is accurate, as the annotator only needs to delete false positive detections and create the missing false negative spans. This functionality can be achieved with a POS tagger built inside OneLabeler that automatically detect numeric values. Moreover, if the developer is not satisfied with the built-in POS tagger, the developer can also implement a customized default labeling module.

Below is the needed code for the developer to customize a default labeling module for text span detection with a noun detector based on POS tagging. The module is implemented in the form of an algorithm server.

```
# A customized default labeling module
# for span annotation by detecting nouns.
import json
import nltk
import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web
import uuid
from typing import List

# Core function
def predict(data_object: dict) -> List[dict]:
    """ Create default spans by detecting nouns. """
    sentence = data_object['content']
    pos_tag = nltk.pos_tag(sentence.split())
    start = 0
    end = 0
    spans = []
    for i, (segment, tag) in enumerate(pos_tag):
        start = end + 1 if i != 0 else end
        end = start + len(segment)
        # filter tags that are not nouns
        if tag not in ['NN', 'NNP', 'NNS']:
            continue
        spans.append({
            'text': segment,
            'start': start,
            'end': end,
```

```
            'category': 'subject',
            'uuid': str(uuid.uuid4()),
        })
    return spans

class DefaultLabelingHandler(tornado.web.RequestHandler):
    def post(self):
        self.set_header('Access-Control-Allow-Origin', '*')
        data_objects = json.loads(self.request.body)['dataObjects']
        labels = [{ 'spans': predict(d) } for d in data_objects]
        self.write({ 'labels': labels })

def main():
    tornado.options.parse_command_line()
    http_server = tornado.httpserver.HTTPServer(
        tornado.web.Application(
            handlers=[(r'/defaultLabels', DefaultLabelingHandler)],
            debug=True,
        )
    )
    http_server.listen(8007)
    print('Serve at http://localhost:8007/defaultLabels')
    tornado.ioloop.IOLoop.instance().start()

if __name__ == "__main__":
    main()
```

Fig. 4.4(A) shows the visually programmed workflow for this labeling system. The resulting workflow of the generated labeling system starts from a random sampling of data objects (**Data Object Selection**). To save the effort of annotating text spans, a machine-learned POS tagger is applied to extract phrases denoting values from the sentence (**Default Labeling**). Within a labeling panel that displays a single sentence, the user can edit its class category (Fig. 4.4(B_1)), use brush to create named entities (Fig. 4.4(B_2)), and link entities to create entity relation annotation (Fig. 4.4(B_3)) (**Interactive Labeling**).

4.5.4 Mixed-Initiative Image Classification

Image classification is a common data labeling application scenario. We demonstrate that OneLabeler allows fast prototyping of an image classification system that features mixed-initiative sampling (with clustering and interactive projection) and default labeling (with decision tree).

To support efficient annotation, aside from semi-automatic labeling mentioned above, cluster-based labeling (e.g., [67, 255]) is also a common technique that involves algorithm and interface design. In the labeling system, the developer may decide to incorporate these two ideas by integrating: (1) default labeling so that annotators only need to do correction (i.e., semi-automatic labeling), and (2) mixed-initiative sampling for selecting similar data objects so that multiple data objects sharing the same label

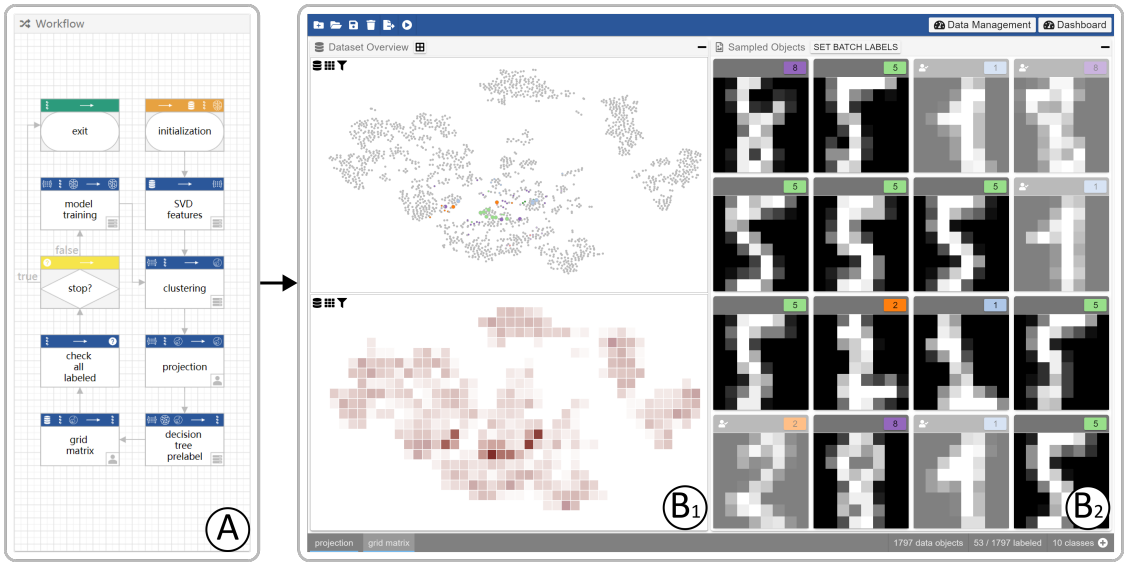


Figure 4.5: Building a mixed-initiative image classification system: (A) User-configured workflow for the system. The interface of the generated image classification system: (B_1) A projection of data object features allowing annotators to use lasso selection for **data object selection**; (B_2) A labeling panel showing images for an annotator to conduct **interactive labeling**.

may be selected together, allowing one label to be assigned to them simultaneously. The mixed-initiative sampling may be accomplished by joining a data object selection module implemented with a clustering algorithm and a projection view that enables annotators to initiate cluster selection [24, 169].

To build a labeling system that incorporates these two ideas, the developer may first start from the simple labeling workflow template shown in Fig. 4.7(A). The developer can further edit the workflow and leverage OneLabeler's built-in interface and algorithm modules for data object selection and default labeling.

To support default labeling, the developer adds a default labeling module and uses model prediction as its implementation. To facilitate model prediction, a model training module is needed where the developer may choose a decision tree as the trained model. Additionally, feature extraction is required as a decision tree is not an end-to-end model, and the developer may choose SVD as the feature extraction implementation. To support mixed-initiative sampling, the developer may first add a data object selection module and choose clustering by k-means as an algorithmic implementation for it so that data objects are sampled cluster by cluster, potentially grouping similar data objects. To allow annotators to involve in the selection process, the developer can add another data object selection module implemented with interactive projection.

Fig. 4.5(A) shows the finalized visually programmed workflow of this labeling system and the resulting labeling system with the UCI handwritten digits dataset [77] loaded.

In this application, the concerned data objects are images of handwritten digits, and the task is to assign digit labels (0, 1, ..., 9) to the images.

The resulting workflow of the generated labeling system starts from singular value decomposition (SVD) as the feature extraction method for processing images (**Feature Extraction**). Mixed-initiative sampling is accomplished by sampling data objects by clustering and then allowing the annotator to revise the samples if needed (**Data Object Selection**). Specifically, a clustering with k-means is run to group and sort data objects where a batch of 16 data objects is sampled each time. The interface displays a projection of the data objects, allowing the annotator to use lasso selection to manually sample data objects (Fig. 4.5(B_1)). A decision tree is used to assign default labels for the sampled data objects by machine and annotator (**Default Labeling**). Within a grid-based labeling panel, the user can edit the labels of sampled data objects (Fig. 4.5(B_2)) (**Interactive Labeling**). Each time the user confirms the current samples are correctly labeled, the interim decision tree model is updated to make the default labeling more accurate (**Model Training**).

4.5.5 Data Extraction from Historical Visualizations



Figure 4.6: Building an interactive machine learning system for visualization data extraction: (A) User-configured workflow for the system. The interface of the generated system for visualization data extraction: (B_1) An overview image that allows the annotator to conduct **interactive labeling** and overview the data extraction result; (B_2) A labeling panel showing the geometric objects for the annotator to conduct **interactive labeling**.

Apart from enhancing conventional labeling processes with algorithm and interface modules, OneLabeler can also support fast prototyping of interactive machine learning systems with customized interactive labeling interfaces. Chapter 3 demonstrates that data extraction tasks can be converted to classification tasks, which are data labeling

tasks. This section demonstrates reproducing the data extraction system demonstrated in Chapter 3 with OneLabeler.

Following the workflow figure of the system (Fig. 3.3), we construct a data labeling workflow to reproduce the system. Note that excluding the preprocessing and post-processing modules, the remaining part of the workflow in Fig. 3.3 can be seen as a workflow for image classification with machine assistance. In this application, the concerned data objects are geometric objects (e.g., polygons), and the task is to assign boolean labels (true detection, false detection) to the objects. The geometric object is not a common data type, and OneLabeler has no built-in support for it. Thus, the developer needs to implement a customized data-type plugin that specifies the import handler and display handler for geometric object instances. In the import handler, the image processing algorithm code developed for the original system in Chapter 3 is evoked for extracting geometric objects from images. In the display handler, we write a simple script for displaying polygons.

Active learning selects informative data objects according to label entropy-based criteria. The label entropy is computed from label distribution estimated by a label propagation model (**Data Object Selection**). A decision tree is used to assign default labels (**Default Labeling**). The user can label geometric objects in the grid-based interface (Fig. 4.6(B_2)) or in the overview image (Fig. 4.6(B_1)). The overview image with extracted polygons embedded is provided to OneLabeler as a plugin for interactive labeling (**Interactive Labeling**). In the model update phase, the label propagation model used for entropy computation and the decision tree model used for default labeling are both updated (**Model Training**).

Fig. 4.6 shows the visually programmed workflow for this system and the interface of the resulting system reproduced with similar features as in the literature. The reproduced system mimics not only the interface design of the original system (an overview plus an annotation panel) but also major algorithm modules for machine-aided labeling (e.g., active learning, algorithmic default labeling). OneLabeler has the potential to accelerate the research on interactive machine learning by allowing researchers to build research prototypes more efficiently. The customization required writing in total around 200 lines of TypeScript code for plugins (excluding the reused image processing code). Specifically, around 50 lines of code are needed to implement the data type plugin to support polygon as a customized data type. Around 150 lines of code are needed to implement the overview image that embeds the extracted polygons. In contrast, the front end of the original system contains around ten thousand lines of code.

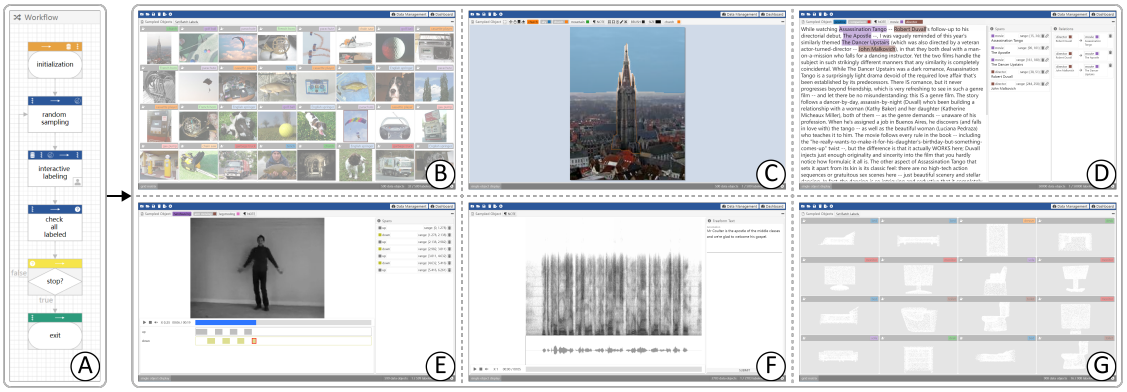


Figure 4.7: A gallery of labeling systems built with OneLabeler using a single workflow (shown in Ⓐ) but different configurations for the interactive labeling module. The configurations differ in data type, label task type, and interface design. The labeling systems ⒸⒹⒺ all support classification, multi-label classification, and freeform text annotation, while additionally supporting label task types that are specialized to the corresponding data type (e.g., object detection for image). (A) A minimal data labeling template that can be used to generate the six labeling systems ⒷⒸⒹⒺⒻⒼ, with some modifications to node configurations for each labeling system; (B) An image classification system; (C) A multi-task image labeling system (additionally support object detection, and segmentation); (D) A multi-task text labeling system (additionally support span tagging, and span relation annotation); (E) A multi-task video labeling system (additionally support span tagging); (F) An audio freeform text annotation system; (G) A point cloud classification system.

4.6 User Study with Developers

We have conducted an in-lab user study to evaluate the usability of OneLabeler for prospective developers of data labeling systems.

4.6.1 Experimental Design

Participants: We recruited eight participants with programming experience between 9 to 12 years and different levels of experience in data labeling. Five (P1, P2, P3, P6, P8) had developed data labeling systems. One (P4) had an ongoing research project that required developing labeling systems. The other two (P5 and P7) had participated in labeling system design and conducted labeling tasks as annotators. The programming expertise of the participants varies. All of them could write back-end or algorithm code using Python or C♯. Three (P1, P2, P8) also had experience in front-end development using JavaScript.

Procedure: The study lasted around two hours for each participant. In each session, we started from asking about the participant's experience in programming and data labeling (~ 15 minutes); next, we gave training on the OneLabeler system (~ 30 minutes); then, we asked the participant to build labeling systems using OneLabeler (~ 50 minutes); finally, we collected feedback on OneLabeler usage experience

(~ 15 minutes).

In training, we showed a documentation website on OneLabeler's usage. The documentation introduces basic concepts in OneLabeler and its visual programming functionalities, including workflow editing interactions, static checking, and the labeling system preview.

We designed the following four tasks for each participant to accomplish with OneLabeler:

- Task 1: Build an image segmentation system similar to that shown in Fig. 4.7(C) without using any built-in template.
- Task 2: Build an image classification system similar to Fig. 4.7(B) based on the workflow built for Task 1.
- Task 3: Build a machine-aided image classification system similar to Fig. 4.5(A) based on the workflow built for Task 2.
- Task 4: Either reproduce the participant's own labeling system or build a webpage classification system similar to that shown in Fig. 4.3.

Table 4.4: Summary of task completion status and time (in minutes).

Participant	Task 1	Time	Task 2	Time	Task 3	Time	Task 4	Time
P1	complete	9	complete	5	with hint (state initialization)	13	complete	12
P2	complete	13	complete	4	complete	17	complete	11
P3	with hint (parameter)	15	with hint (parameter)	4	complete	12	partially complete	14
P4	complete	29	complete	5	with hint (linting message)	25	complete	4
P5	complete	16	complete	3	complete	14	partially complete	12
P6	with hint (parameter)	15	complete	9	with hint (state initialization)	25	partially complete	22
P7	complete	9	complete	5	complete	18	complete	4
P8	complete	8	complete	8	complete	13	complete	11

We designed the first three tasks to evaluate the overall OneLabeler usability. Specifically, Task 1 evaluates how a user builds new workflows from scratch, which reflects the understanding of basic concepts and usage of OneLabeler. Task 2 evaluates how a user adapts an existing workflow to a new usage scenario. Task 3 evaluates how a user builds complex bespoke workflows. Moreover, we designed Task 4 to further understand if OneLabeler can be used to build a custom labeling system for real-world usage. It is an open-ended task. A participant could either reproduce a labeling system the participant had built before or extend OneLabeler with custom modules to fulfill the desired labeling scenario. For each task, we provided the participants with a textual specification of the desired functionality of the labeling system (see Appendix B).

4.6.2 Results

Task Completion

Table 4.4 summarizes the task completion status and time for the four tasks. For the first three tasks (T1, T2, T3), all the participants were able to complete them. Out

of the 24 trials in total (8 participants × 3 tasks), the participants could finish the tasks correctly in 18 trials without hints from the experimenter. Four participants (P1, P3, P4, P6) needed hints in six trials to make their results exactly the same as the specification given in the instructions.

We observe that front-end development experience may influence the speed and the need for hints, even for tasks that do not require textual programming. For the first three close-ended tasks, the three participants (P1, P2, P8) with front-end development experience spent on average about 30 minutes and needed in total 1 hint, while the other five participants (P3, P4, P5, P6, P7) spent on average about 41 minutes and needed on average 1 hint per person. While our number of participants may be insufficient to compare the performance between participant groups reliably, we hypothesize that our toolkit might be easier to learn and faster to use for developers with front-end development experience. A possible reason is that our visual programming interface is web-based, and developers with front-end developers have more experience using web-based interfaces during their previous development experience.

For Task 4, three of the participants (P4, P6, P7) chose to reproduce their own labeling systems: both P4 and P7 developed a text labeling system similar to Fig. 4.7(D), and P6 built a system for pairwise comparison of images. The other five participants selected the alternative to build a predefined webpage classification system. Five of the eight participants (P1, P2, P4, P7, P8) completed Task 4 independently. The other three participants (P3, P5, P6) completed Task 4 after receiving help from the experimenter. The experimenter helped P6 with the coding part, as P6's system required creating a new data type for image pairs, while P6 was unfamiliar with the web development skills needed for the customization. P3 and P5 also had no web development experience. As a result, they could not complete the coding part when building the webpage classification system. After the experimenter helped them develop the required customized module, they constructed the workflow to achieve the desired labeling system.

Usability

As new users, all the participants could use OneLabeler to quickly accomplish the assigned tasks to build labeling systems. This indicates that OneLabeler is easy to learn and easy to use overall. For most participants, building a new workflow from scratch (Task 1) took less than 16 minutes, adapting an existing workflow to a new usage scenario without adding or removing a node (Task 2) took less than 10 minutes, and adding machine assistance to an existing workflow (Task 3) took around 15 minutes. For Task 4, which requires coding, participants with web development experience (P1, P2, P8) accomplished the task independently and efficiently within 15 minutes. In the interview, all the participants commented that OneLabeler was flexible and comprehensive.

During the tasks, all the participants utilized the static checking function to resolve occurred issues, indicating the usefulness of static checking. During the tasks, participants looked up the documentation website several times. For example, P1, P2, and P6 looked up the definition of the conceptual modules, while P5 and P7 looked up the detailed illustration for the error code "no-uninitialized-inputs" (the Input Initialized rule in Section 4.3.5). It indicates that the documentation website provides helpful information and that OneLabeler requires some learning.

Potential Improvements

Through the user study, we have identified several opportunities to improve OneLabeler's usability. The completion time of the tasks was affected by misoperations. The fastest participant (P8) spent only 8 minutes on Task 1, while P4 spent 29 minutes on the same task. The reason for P4's lengthened completion time in Task 1 was that P4 clicked the browser's Refresh button by mistake, which caused the previous progress to be lost, and thus P4 had to rebuild the workflow. Similarly, P6 refreshed the webpage and spent 25 minutes on Task 3. This indicates a potential usability improvement by supporting automatic saving of user progress. Additionally, P1 suggested that OneLabeler should provide a function for refining the workflow layout with one click to save the developer's time manually arranging nodes.

In the first three tasks, some participants needed hints from the experimenter to make their workflows consistent with the specification. Three of the hints (in Task 1 for P3, Task 1 for P6, and Task 2 for P3) were given due to wrong parameter choices in their workflows. For example, the initial workflows created by both P3 and P6 in Task 1 used the default value 48 for the number of data objects sampled by random sampling instead of setting it to 1 as required in the textual specification of the task. These mistakes were caused by carelessness as they forgot to change the configuration from the default value. In real-world usages, these mistakes caused by carelessness may be less common, as the specifications for real-world usages are driven by the developer's own needs, and such mistakes would be easy for the developer to spot. Additionally, we may explicitly remind the user to check whether the configuration needs to be changed from default values by providing a suppressible warning message for each component using the default configuration. It may also be beneficial to reduce such mistakes by improving the documentation to stress the necessity to try out the labeling system preview and carefully check if it functions as expected.

The other three hints (in Task 3 for P1, P4, and P6) were given as the participants did not recognize that the "model" state needs to be initialized before use. In this case, OneLabeler's static checking provided multiple recommended fixes, as "model" can be initialized either by declaring it as an output of the initialization node (described in Section 4.4.2) or by adding a module that outputs model. The participants either took a wrong path to fix it by adding unnecessary modules, deviating from the specification

(P4), or recognized that they needed to initialize the state but did not realize that the initialization could be done in the initialization node and got stuck (P1 and P6). This suggests further improvements to provide more understandable error messages to help developers locate and fix errors more effectively.

Future Usage Scenarios

According to the participants, existing data labeling systems usually are not suitable for their own tasks, mainly due to the following three reasons:

1. **Unconventional labeling tasks**: For example, P1 needed to annotate visual objects with their attributes in visualizations stored as vector and bitmap images. There exist no mature labeling systems for this labeling task.

2. **Requirement of algorithm support**: To save annotation costs, a labeling system may integrate algorithm support. The suitable algorithm largely depends on the usage scenario. For example, P3 developed an auto-tracking function to infer bounding boxes of data objects given the bounding box annotations of previous frames. While P3 found an existing labeling system that provides basic auto-tracking support, P3 could not adapt it to meet the requirement, because it was closed source.

3. **More control over software changes**: The data labeling requirements can change over time. For example, P2 said *"Even if there exists a piece of software that temporarily meets my requirements, I would be diffident when using it. I may need to abandon it at some point when the requirements can no longer be accommodated by the existing software."*

All the participants agreed that OneLabeler could fill these gaps with its built-in features and customization support that enable efficient building and low-level control of the implementation.

4.7 Discussion

OneLabeler is an attempt to improve the efficiency of building data labeling systems, especially intelligent labeling systems that integrate algorithmic approaches to assist annotators. The design of OneLabeler follows the rationale that accelerating the building requires promoting the reuse of software modules, which in turn requires identifying commonalities in existing data labeling systems. Following this rationale, OneLabeler builds on a system architecture based on common data labeling modules in the literature.

On the expressiveness of conceptual modules: Although there is no guarantee that the primitive modules (i.e., the API definitions) in OneLabeler are capable of

representing all possible variations in data labeling systems, there is evidence that they are expressive enough to cover a wide range of interesting variations. Firstly, the coding process that leads to the modules provides evidence that the modules are expressive enough, as they conceptually can represent at least the labeling-related modules in the 33 related papers. Moreover, the usage scenarios provide additional evidence that OneLabeler can generate diverse labeling systems in action.

Build in more module implementations: A clear direction to enhance OneLabeler is to extend it with more built-in modules. We aim to provide more built-in options for the lower frequency modules currently underexplored (e.g., quality assurance, stoppage analysis, and label ideation). Especially, while quality assurance is not frequently mentioned in the papers we collected, we expect it to be important in real-world applications. Additionally, while the conceptual modules of OneLabeler are sufficient to produce diverse data labeling systems, they may not capture all common techniques related to data labeling emerging in all the related fields. Meanwhile, the coding process presented in Section 4.3 may be reapplied to a larger corpus of literature to improve the coverage of conceptual modules. Through this iterative process of extending the expressive power, we move gradually towards the ultimate goal of making OneLabeler *feature complete*, with various built-in implementations (algorithms or interfaces) available for multifarious real-world usages.

On orthogonality of conceptual modules: While OneLabeler builds on primitive modules, the conceptual modules in the proposed framework are not strictly orthogonal. For example, both the quality assurance module and the interactive labeling module output labels. OneLabeler does not and should not forbid developers to use an implementation of quality assurance for the interactive labeling purpose. We believe that enforcing strict orthogonality in the API definitions is unnecessary. For example, if the interactive labeling module and the quality assurance module were merged into a single module for strict orthogonality, the discoverability of interactive labeling implementations and quality assurance implementations would both decrease. A developer who wants to configure the module into an implementation of interactive labeling would have to pick the implementation out of all the interactive labeling implementations and quality assurance implementations, which is cumbersome.

On discoverability of modules: As the number of built-in module implementations increases, the options provided by OneLabeler may become too many for novice users to discover. We address this potential issue by allowing new implementations plugged into OneLabeler and built-in modules deleted from OneLabeler, as each module implementation in OneLabeler's source code is an individual script file. In this way, we can provide multiple versions of customized distributions of OneLabeler, each with a different set of built-in modules suited for different developers. In addition, OneLabeler can provide advanced search and recommendation functionalities in the module configuration panel to alleviate this issue in the future.

Extension to crowd labeling scenarios: In this chapter, we have not explicitly considered the usage scenario of crowd labeling workflows. Our framework may accommodate some aspects of crowd labeling workflows, e.g., the task scheme design in crowdsourcing [168] is related to the task design in interactive labeling and postprocessing of crowdsourced labels is related to quality assurance. However, others are not captured by the conceptual framework. To support crowdsourcing, OneLabeler needs to build in additional interface and algorithm modules, such as for administrator's tasks, including monitoring label progress, assessing annotators' reliability, and data management. OneLabeler's states also need to be extended to accommodate new modules' inputs and outputs. To integrate with existing infrastructures, we also need to investigate how to distribute a labeling system created with OneLabeler on crowdsourcing platforms.

Exploring other usages of OneLabeler: While OneLabeler is designed for building labeling systems, as demonstrated in Section 4.5, we envision that OneLabeler may be extended to support interactive machine learning applications other than data labeling. For example, OneLabeler may be extended to interactive model training if additional interface modules are added to OneLabeler to allow end users to edit the model. Another example is to apply OneLabeler to build customized information retrieval systems (e.g., photo management systems) by adding more modules for data object selection with information retrieval techniques. Additionally, OneLabeler is designed for building standalone labeling systems, while it may also be beneficial to explore building UI widgets for data labeling that can embed on other systems.

On Customization: Customization support is essential to ensure the flexibility of OneLabeler, as revealed by Task 4 in the user study. OneLabeler allows a developer to add customized implementations, as long as the inputs and outputs of the implementations are within OneLabeler's supported states. Meanwhile, the customizability of OneLabeler may be extended in future work to support customization on workflow execution and UI appearance. Fine control on parallelizing execution of multiple modules, such as waiting for all or racing, cannot be achieved at the moment. Besides, each interface module currently appears as a window in OneLabeler's interface, and UI widgets outside a window are not customizable. As customization requires textual programming, OneLabeler provides CLI (Section 4.4.4) to assist customization development, where a developer can start coding based on template code rather than from scratch. To further reduce the textual programming required for customization, it is beneficial to build a marketplace for OneLabeler modules, where developers can share their customized modules for others to reuse. It may also be beneficial to further decompose common modules into submodules. Through subdivision, the flexibility for reuse increases, as a developer may customize a submodule instead of an entire module.

4.8 Conclusion

This Chapter has proposed a conceptual framework for data labeling and the OneLabeler toolkit based on the conceptual framework to support easy building of intelligent labeling systems for diverse usage scenarios. To build the framework, we have identified common states and modules by coding the literature and summarized constraints in composing the modules to build labeling systems. Each modular process can be instantiated as a human, machine, or mixed computation procedure. OneLabeler provides a visual programming interface. It builds in various implementations of modules in the conceptual framework as building blocks for reuse. Developers can create labeling systems with no code or less code. It provides static checking and preview functionalities to assist development and debugging. OneLabeler supports customization, allowing developers to extend its capability further.

We demonstrated OneLabeler's expressiveness through usage scenarios of building ten data labeling systems, including a reproduction of the data extraction system in Chapter 3. We have further conducted a user study to evaluate its usability and collect feedback from potential users. The user-study results suggest that OneLabeler is easy to learn and enables potential users to build labeling systems efficiently.

Chapter 3 demonstrates that a data extraction problem can be converted into a series of classification tasks, which are instances of data labeling tasks. The conversion connected data extraction and data labeling. From the perspective of the end-users assigning labels in the block detection, block grouping, and root detection tasks introduced in Section 3.4, they are essentially assigning image classification labels. This connection implies that our approach for saving user effort in data extraction, particularly active sampling and default labeling, can be used to save user effort in data labeling. Similarly, various techniques in data labeling can also be adapted to accelerate data extraction.

Section 4.5.5 demonstrate that OneLabeler can be used to develop data extraction systems. Fundamentally, for the generic workflow introduced in Fig. 3.3, aside from the preprocessing and postprocessing modules that are highly dependent on the data extraction workflow, the other modules are typically reusable between workflows. The cost of implementing these modules, such as active sampling and default labeling, can be reduced with OneLabeler.

Note that Section 4.3 has also summarized reusable implementations in data extraction systems but focuses on high-level building blocks. The building blocks synthesized in this Chapter are low-level and can be used to implement the high-level building blocks in Section 4.3.

"All models are wrong, but some are useful."

— George E. P. Box

5 Model-Based Evaluation of Label Quality Assurance Interfaces

5.1 Introduction

Machine learning (ML) models rarely achieve 100% prediction accuracy in real-world applications. In some applications, such as search engines and product recommendations, the end-users can ignore prediction errors with little cost, and thus the errors are tolerable. However, prediction errors cannot be ignored in many other applications, such as medical diagnosis, fraud detection, and content moderation. Such applications are characterized as *quality-sensitive*, where rigorous quality assurance (QA) of model predictions is necessary.

Data extraction from historical visualizations is one such scenario. Errors in the extracted dataset may prohibit further usage of the dataset. Section 3.2 involves quality assurance as a step of data extraction. Generally, quality assurance is essential to ensure the quality of data labels in data labeling systems. Section 4.3.4 identifies quality assurance as a common type of modules in data labeling systems.

During QA processes, human needs to review machine predictions and correct

The research in this chapter is published as "Yu Zhang, Martijn Tennekes, Tim de Jong, Lyana Curier, Bob Coecke, and Min Chen. 2021. *Using Simulation to Aid the Design and Optimization of Intelligent User Interfaces for Quality Assurance Processes in Machine Learning.* arXiv: 2104.01129 [cs.HC]." [298] and "Yu Zhang, Bob Coecke, and Min Chen. 2019. *On the Cost of Interactions in Interactive Visual Machine Learning.* In Proceedings of the IEEE VIS Workshop on Evaluation of Interactive Visual Machine Learning Systems. Vancouver, BC, Canada. IEEE." [296]. The overall idea was developed by Yu Zhang and Min Chen through discussions. Yu Zhang conducted the implementation, experimentation, and writing. Min Chen, Bob Coecke, and Tim de Jong guided the work. Tim de Jong, Martijn Tennekes, and Lyana Curier provided domain knowledge on remote sensing. Min Chen revised the writing. This work is conducted in collaboration with Statistic Netherlands. It is supported by European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 822214.

erroneous predictions in a user interface (UI). This chapter focuses on interfaces for performing such repetitive tasks of quality-assuring ML model predictions. We refer to these interfaces as QA interfaces.

In the software development life-cycle of an interactive system, it is important to explore and evaluate many different design options to select a relatively optimized design to be implemented, tested, and deployed. The design optimization process can also be applied iteratively in the life-cycle to improve a user interface. Since most ML models are developed to handle big data, a QA interface is typically used in conjunction with a huge dataset. With such an interface, users often have to spend much time performing laborious operations of viewing data objects and machine predictions and correcting errors when prediction errors are identified. For such labor-intensive QA processes, it is desirable to evaluate a diverse range of interface variants to identify the design option that maximizes the user's productivity. Due to the labor-intensive nature of QA, this chapter focuses on the evaluation metric of *task completion time* in QA.

It is common to adopt a user-centered design process [31, 269] to design a user interface following a requirements analysis phase. After the software is developed, one conducts usability studies to evaluate the user interface. Such user studies can inform the improvement of the user interface at different stages of the software life-cycle [54, 278]. While opinions of potential users and usability studies can assist the design of such interfaces, these studies are usually costly and time-consuming as they intensively involve human subjects. It is difficult to evaluate a large number of design options while taking into account various factors, such as users' skills and strategies in performing a large collection of operations. It is also difficult to conduct user studies during the design phase, i.e., before a user interface has been developed.

This chapter proposes a model-based evaluation approach to efficiently evaluate QA interfaces by modeling and simulating the task completion time (Section 1.2 **O3**). We specifically focus on grid-based interfaces for QA, which we have used in our data extraction (in Section 3.4.4) and data labeling systems (in Section 3.4.4). Our approach is based on a model of the user's routine operations in QA processes. Our approach can be seen as an adaptation of the Keystroke-Level Model (KLM) to estimate quality assurance tasks' completion time in grid-based interfaces [48].

Using simulation, UI specialists (e.g., designers, developers, or evaluators) can explore design options at a scale unattainable with user studies. We demonstrate applying this approach to evaluate QA interfaces in data extraction and aerial image classification systems.

We model and simulate the influence of various factors on the QA time cost. In this chapter, we describe the model in a manner of increasing its complexity gradually. In Section 5.3, we describe our simulation approach, overview the simulation model, and outline the factors to be modeled and simulated. We start with modeling simple QA interfaces that only provide single edit commands. Section 5.4 focuses on the factors of

interface layout and application scenarios. The simple modeling exercise demonstrates the feasibility of using simulation to guide the optimization of interface layout. The availability of interaction functions in QA interfaces and algorithmic modules may affect the time cost. Batch edit functions enable users to edit the label of multiple data objects simultaneously. Section 5.5 investigates the influence of introducing batch edit functions. After adding the batch edit commands, users may accomplish QA tasks with different combinations of single or batch edit commands. In Section 5.5, we introduce the factors about different user strategies in combining different commands. The accuracy of the predictions made by the ML model is a factor that influences the overall QA time cost. In Section 5.6, we investigate the modeling of the impact of this factor. While QA interfaces may randomly present data objects to users, intelligently ordering data objects can enable more uses of batch edit commands. In Section 5.7, we bring in the factor of algorithmic rank functions for ordering data objects into the simulation model. In Section 5.8, we summarize the overall simulation model with all the factors integrated together.

The main contributions of this chapter include:

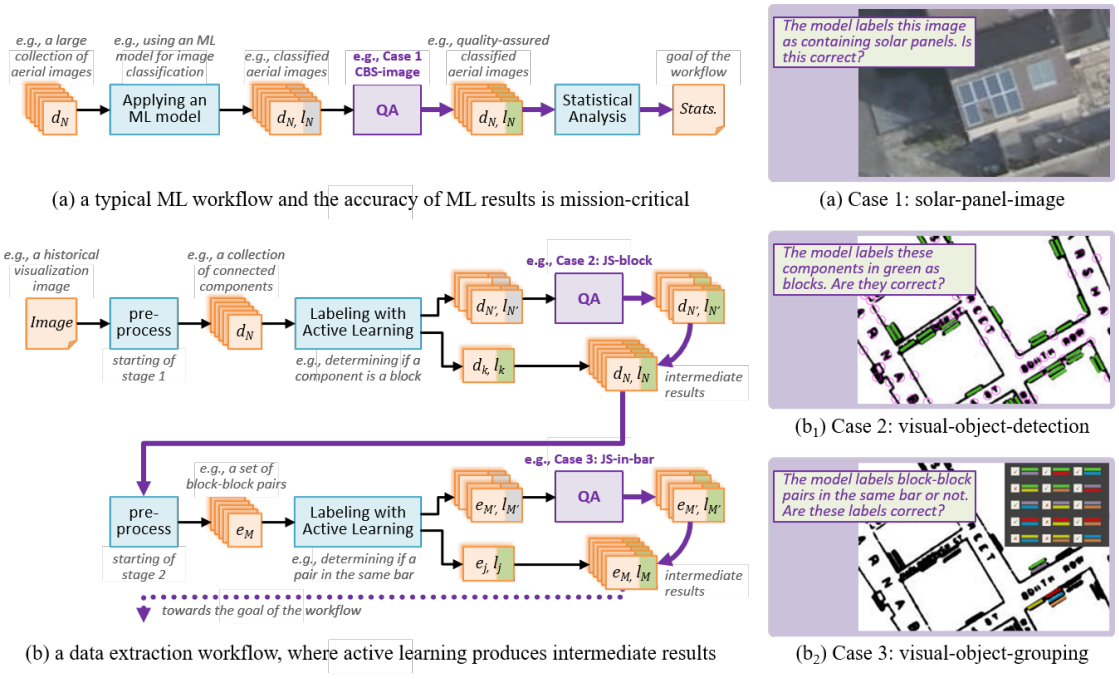
- a simulation **model** of users' routine operation sequence and operation time costs in quality assurance interfaces, and
- an **application** of the model to evaluate and optimize quality assurance interfaces for data extraction and aerial image classification.

5.2 Background

This section introduces two example data-centric applications and their QA interfaces to illustrate the need for QA. Then, we present a template grid-based interface that typifies the QA interfaces.

5.2.1 Data-Centric Applications

Visualization Data Extraction: The first application concerns detecting and grouping visual objects in historical visualizations as introduced in Chapter 3. For example, given John Snow's famous cholera map [250], the rectangular blocks are to be detected (Fig. 5.1(b_1)) and grouped (Fig. 5.1(b_2)). The detected visual objects are then used to extract the underlying datasets generating the visualizations. While machine-learned models may not accurately detect the visual objects, they can suggest visual objects and tentative classification labels for the user to quality-assure, as demonstrated in the MI3 workflows in Section 3.4. In this application, QA is needed for the correctness of the extracted datasets.



(a) a typical ML workflow and the accuracy of ML results is mission-critical

(a) Case 1: solar-panel-image

(b) a data extraction workflow, where active learning produces intermediate results

(b_1) Case 2: visual-object-detection

(b_2) Case 3: visual-object-grouping

Figure 5.1: ML workflows in data-centric applications requiring quality assurance: (a) A workflow to generate summary statistics from the classification of whether an image contains solar panels. Quality assurance of classification labels is needed to achieve reliable statistics. Following the terminology in Fig. 3.3 and Fig. 4.1, the step of "Applying an ML model" is a default labeling module, and "QA" refers to a quality assurance module. (b) A workflow with two stages (b_1 and b_2) to extract datasets from visualization images. Quality assurance is necessary to avoid propagating errors to the succeeding stage and guarantee the correctness of the extracted dataset. This workflow is a redraw of Fig. 3.3 where all the nodes in the active learning loop are wrapped into "Labeling with Active Learning".

Solar Panel Detection: The second application concerns generating a labeled benchmark image dataset where labels classify aerial images as containing solar panels or not (Fig. 5.1(a)). The benchmark dataset is used to generate official statistics on solar panel deployment. In this application, while using ML models' predictions may accelerate the labeling, predictions not checked manually are untrustworthy and imprecise. A QA process inspecting and correcting machine predictions is necessary for the correctness of the official statistics. Fig. 5.1(a) shows this process of applying an ML model and quality-assuring machine predictions to generate a benchmark dataset and, consequently, official statistics.

These applications typify scenarios of carrying out real-world *data-centric* ML projects. In these scenarios, the ultimate goal is to gather high-quality labeled/processed data. Using ML in these projects is not to train accurate models but to assist the data labeling and processing. These applications exhibit a common challenge that a user may need to perform many tedious quality assurance operations to correct machine prediction errors. It is critical to optimize QA interfaces to reduce user

operation costs.

5.2.2 Grid-Based Interfaces

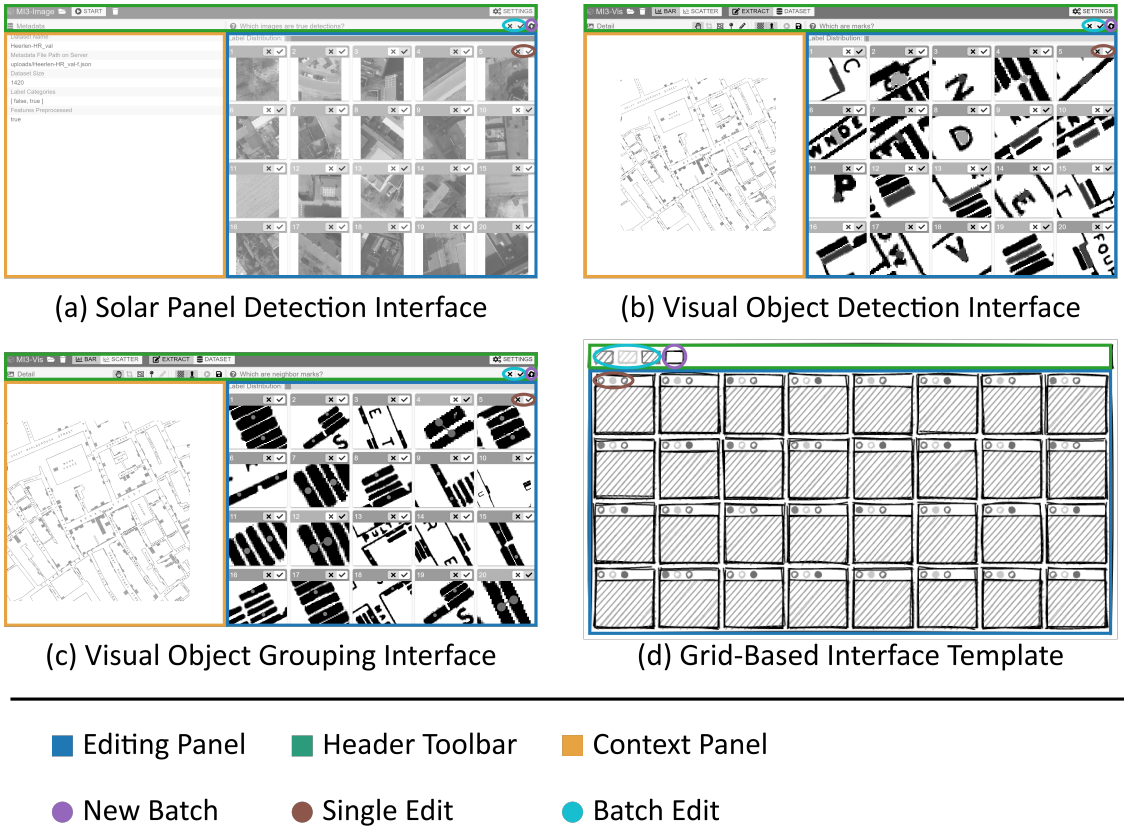


Figure 5.2: Examples of QA interfaces: The interface components (editing panel, header toolbar, and context panel) are annotated with colored rectangles. The buttons corresponding to operations (new batch, single edit, batch edit) available to the user are annotated with colored ovals. The interfaces all contain an edit panel that presents a matrix of thumbnails showing data objects, a header toolbar providing interaction functions, and a context panel that provides an overview of the dataset: (a) An interface for solar panel detection corresponding to workflow in Fig. 5.1(a). (b) An interface for visual object detection corresponding to workflow in Fig. 5.1(b_1). (c) An interface for visual object grouping corresponding to workflow in Fig. 5.1(b_2). (d) The typical design of grid-based interfaces in QA.

QA requires users to verify machine predictions in interfaces. Fig. 5.2 shows three QA interfaces corresponding to the two applications described above. Fig. 5.2(a) shows an interface for classifying solar panel detection. In this interface, an ML model is used to assign tentative classification labels for the image on whether they contain solar panels. Fig. 5.2(b, c) show interfaces for classification for visualizations data extraction introduced in Section 3.4.4. The interface in Fig. 5.2(b) displays a batch of visual objects. The visual objects are to be categorized as encoding data (in green) or not (in orange). The context panel on the left shows part of John Snow's cholera map under

check. In these interfaces, ML models are used to extract and group visual objects.

The quality assurance interface can be divided into three parts: the editing panel, header toolbar, and context panel. The editing panel is where a user performs most operations to correct ML prediction errors. Because data objects to be inspected are usually too many to be quality-assured all at once, the editing panel normally shows a subset (referred to as a "batch") of data objects each time. The header toolbar may provide utilities such as dataset upload, requesting the system to present the next batch of data objects to be quality-assured, and simultaneously setting labels for a batch of data objects. For example, in Fig. 5.2(a - c), each interface shows a batch of 20 data objects. The context panel gives the user a dataset overview and may present summary statistics about the labeling progress, such as the machine errors identified. The following mainly focuses on the editing panel and header toolbar as they accommodate the functions directly related to quality assurance.

In all the interfaces, users need to manually verify machine predictions in a grid panel on the right part of the interface. The grid panel presented each data object as a thumbnail image in a grid cell. Each grid cell provides buttons for editing its label. Fig. 5.2(d) typifies the design of grid-based interfaces. This chapter focuses on grid-based interfaces in QA. We refer to each thumbnail corresponding to a data object and an interaction mechanism (e.g., buttons for editing label categories) as a *grid cell*, and all grid cells in the editing panel as a *batch*. It is the most commonly adopted design in QA interfaces as discussed in Section 4.4.3 with numerous instances in the literature [22, 121, 176, 280, 297].

We list typical functionalities of QA interfaces exhibited in the literature. The interface enables a user to **request a new batch** of data objects to be quality-assured in the editing panel. An algorithmic process (e.g., active learning or clustering) may determine the priority for data objects to be presented to the user. An ML model assigns default labels to the data objects. The user has to inspect data objects to judge whether default labels are correct. The interface provides **single edit** commands in the editing panel for correcting individual label errors. To improve efficiency, the interface provides the user with **batch edit** commands for changing the labels of a group of data objects (e.g., set all labels to positive).

QA interfaces utilize human intelligence to:

- Recognize ML model's prediction errors.
- Determine whether using a batch edit command is more efficient than applying single edit commands to data objects individually.

Batch editing is beneficial when the batch of data objects mostly shares the same label. Consider a batch of 20 data objects with binary labels (e.g., yes or no). If there are 10 true positive labels, 2 false positive labels, and 8 false negative labels, the user

can use a batch edit command to set all labels to positives first and then change the 2 false positive labels to negatives. This process requires 3 user operations. If the user only uses single edit commands, 10 operations are needed.

It also utilizes machine assistance to:

- Assign default labels to data objects.
- Determine the ordering for data objects to be labeled.

Suitable ordering may accelerate the quality assurance process. Firstly, the ordering algorithm may bunch together data objects of the same label, providing more opportunities for applying a batch edit command. Secondly, when the ML model for assigning default labels is incrementally updated with the user corrections, ordering generated by active learning methods may enable the ML model to learn more efficiently, leading to better default labels.

While human-in-the-loop is indispensable for QA processes, it is also essential to evaluate and optimize QA interfaces to reduce user effort. Evaluating different design options of QA interfaces with user-centered evaluation methods (e.g., surveys, small group discussions, controlled experiments) is time-consuming to both designers and potential users.

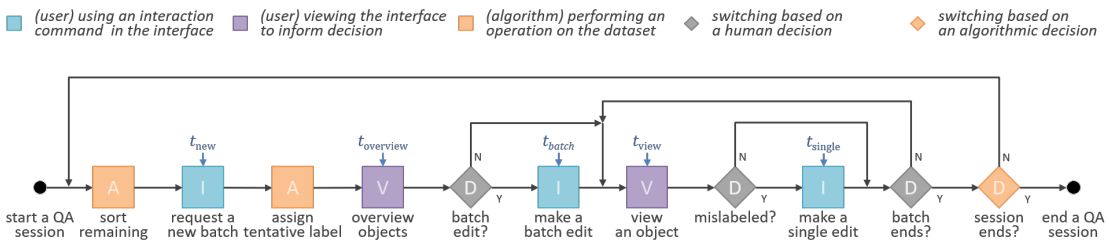


Figure 5.3: A general workflow of user and machine routine operations in a quality assurance session in a grid-based interface typified in Fig. 5.2(d). The operations in the workflow can be categorized into five categories: user interaction in the interface, the user's viewing action, algorithmic operation, user decision, and algorithmic decision.

5.3 Overview of the Simulation Approach

This section outlines a model-based approach to evaluating quality assurance interfaces. We introduce a general workflow capturing user's routine operations in grid-based interfaces for QA. We describe modules in the workflow to be modeled and simulated. The workflow is constructed by analyzing user and machine operations in the QA interfaces typified in Fig. 5.2(d).

Given the workflow, we can estimate the time cost of a QA session by simulating sequences of user and machine operations and adding up the time cost of each

operation. Section 5.3.2 introduces a simulation model for generating operation sequences and adding up the time cost of operations. Section 5.3.3 outlines the factors that we model and simulate. Section 5.3.4 clarifies the scope and assumptions in our simulation-based evaluation.

5.3.1 A Routine Task Workflow

Problem Formulation

In general, quality assurance concerns the following setup. A dataset $X = \{x_i | i = 1, 2, \ldots, n\}$ contains n data objects to be assigned quality-assured labels. An ML model is applied to the dataset X, generating in default labeled $Y = \{y_i | i = 1, 2, \ldots, n\}$ where y_i is the corresponding default label for data object x_i. The user needs to quality-assure the default labels to generate verified labels $Y^* = \{y_i^* | i = 1, 2, \ldots, n\}$.

We refer to the entire process of carrying out operations to quality-assure the n data objects as a *session*. In typical QA interfaces, a subset of data objects, $X_i \subseteq X$, is loaded into the editing panel. The editing panel can accommodate maximal n_{batch} data objects, and a subset of data objects loaded to the editing panel is of size K ($K \leq n_{batch}$). We refer to the process of quality-assuring the subset X_i as a *round*.

For classification tasks, let c be the number of label categories. Let $cm \in [0,1]_{c \times c}$ be the normalized confusion matrix with $\sum_{i,j} cm_{i,j} = 1$. $cm_{i,j}$ denotes the rate of the number of data objects with true label i and default label j over the number of all the data objects (n).

For binary classification, the correctness of each label y_i can be represented with one of the four values: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Let n_{TP}, n_{TN}, n_{FP}, n_{FN} be the number of TP, TN, FP, and FN in the whole dataset. The normalized confusion matrix can be represented as $cm = \frac{1}{n}\begin{pmatrix} n_{TN} & n_{FP} \\ n_{FN} & n_{TP} \end{pmatrix}$.

Operators in the Workflow

Section 5.2.2 introduces user operations and machine operations in typical grid-based interfaces for QA. Using the user and machine operations, we construct a workflow as shown in Fig. 5.3 to summarize procedures in QA. The quality assurance workflow includes two nested loops. The inner loop is between the operations “view an object” and “batch ends?”. Going through the inner loop once quality-assures the label of a data object. The outer loop is between the operations “sorting remaining” and “session ends?”. Going through the outer loop once quality-assures the labels of a batch of data objects. In the following, we introduce each operation in the workflow.

The workflow involves a set of interaction commands, or operators, initiated by the user (shown as blue rectangles in Fig. 5.3), including:

- **Request a new batch** of data objects to quality-assure by activating a "new batch" command in the interface (denote its average time cost as t_{new}).
- **Make a batch edit** to assign the same label to the current batch of data objects by activating a "batch edit" command in the interface (time cost t_{batch}).
- **Make a single edit** to edit the label of a data object in the current batch by activating a "single edit" command in the interface (time cost t_{single}).

Some of the interaction commands can only be issued after the user makes explicit decisions (shown as gray diamonds in Fig. 5.3), including:

- **Batch Edit?** The decision of whether a "batch edit" should be issued, which depends on whether using a batch edit saves effort over only using single edits.
- **Mislabeled?** The decision of whether a "single edit" should be issued, which depends on whether the data object is mislabeled.
- **Batch Ends?** The decision of whether all the data objects in the current batch are quality-assured.

The user needs to perform viewing actions to obtain the information of the interface state to inform some of the decisions (shown as purple rectangles in Fig. 5.3), including:

- **Overview objects** in the current batch and comprehend the overall label category distribution in the batch (time cost $t_{overview}$).
- **View an object** in the current batch and comprehend its label category and whether it is mislabeled (time cost t_{view}).

The machine needs to perform algorithmic operations, including:

- **Sort remaining** to reorder remaining data objects to be quality-assured. In a QA interface, the sorting can be implemented with any algorithm that serves the reordering purpose. For example, the rank method can be implemented with a clustering algorithm to group data objects of the same label category [67, 253, 255]. In this case, the rank method may create good opportunities to use batch edit commands. The sorting may also be implemented by scoring functions in active learning [243] to prioritize data objects whose labels are more uncertain.
- **Assign default labels** to the batch of selected data objects. In the scenario of quality-assuring ML predictions, default labels may be assigned by a pre-trained machine learning model or an incrementally updated model [15, 42, 101]. In general quality assurance scenarios, default labels may come from other sources, such as another user or pre-defined rules.

An algorithmic decision (**session ends?**) determines whether the session should end, which depends on whether all the data objects are quality-assured (shown as an orange diamond in Fig. 5.3).

The original Keystroke-Level Model decomposes the interaction tasks into the keystroke-level operators, such as keystroking, pointing, and mental preparation for physical actions, and assumes each type of operator takes the same amount of time to carry out. As we focus on the modeling of task completion time of quality assurance tasks in grid-based interfaces, we decide not to assume each type of keystroke-level operator to take the same amount of time to carry out, which may lead to more accurate time cost estimation. For example, we do not assume the time cost of "batch edit" (t_{batch}) and "single edit" (t_{single}) to cost the same amount of time, considering that the buttons for batch edit and single edit are distributed at different places in the interface, although they are both a pointing followed by a clicking in the Keystroke-Level Model.

5.3.2 A Simulation Model

Algorithm 3 QA Session Simulation

Input: $n, n_{batch}, \vec{t}$
Output: time cost of the quality assurance session $T_{session}$

$X, Y^* \leftarrow CreateMockDataset(n)$
$ops \leftarrow []$ ▷ log of operation sequence
do
  $X, Y \leftarrow Rank(X, Y)$ ▷ sorting remaining
  ops.push("new") ▷ request a new batch
  $X_s \leftarrow X.slice(n_{batch})$
  $Y_s^* \leftarrow Y^*.slice(n_{batch})$
  $Y_s \leftarrow DefaultLabel(X_s)$ ▷ assign default label
  ops.push("overview") ▷ overview objects
  $cmd \leftarrow SelectEditCommand(Y_s, Y_s^*)$
  if $cmd \in BatchEditCommands$ **then** ▷ batch edit?
    ops.push(cmd) ▷ make a batch edit
    $Y_s \leftarrow ApplyBatchEdit(cmd, Y_s)$
  end if
  for $j \in (1, ..., Y_s.length)$ **do** ▷ loop until batch ends
    ops.push("view") ▷ view an object
    if $Y_s[j] \neq Y_s^*[j]$ **then** ▷ mislabeled?
      ops.push("single") ▷ make a single edit
    end if
  end for
while $Y \neq []$ ▷ session ends?
$T_{session} \leftarrow GetCost(ops, \vec{t})$ ▷ compute total time cost
return $T_{session}$

The workflow provides the basis of a simulation model and defines the top-level

structure of a simulator implementing the model. Algorithm 3 illustrates an implementation of a simulator for QA sessions. The simulation algorithm generates a mock dataset, synthesizes a sequence of operations for quality-assuring the mock dataset, and adds up the time cost of the operations.

The simulation algorithm takes as input the number of data objects n, the number of data objects presented simultaneously in the editing panel n_{batch}, and operator time costs of the operations in the workflow $\vec{t}$. It outputs the time cost of the QA session.

Algorithm 3 is generic in that different implementations of routines such as "CreateMockDataset", "Rank" and "DefaultLabel" can be plugged in to simulate different QA interface designs. For example, at the beginning of the simulation (line 1 of Algorithm 3), a mock dataset is generated by "CreateMockDataset". This routine may also fetch real data objects as we adopt in the simulations in Section 5.4. It may generate a mock dataset according to n and an additional parameter of default labels' confusion matrix, as we adopt in the simulations in Sections 5.5, 5.6, and 5.7. Sections 5.4, 5.5, 5.6, and 5.7 introduce implementations of the routines.

5.3.3 Modeled and Simulated Factors

This section outlines factors of QA interfaces that may affect the time cost of the QA process. These factors inform our modeling and simulation experiments in the following sections.

- **Interface layout**: The grid-based interface may utilize different layouts (as shown in Fig. 5.5). The display size of each data object is affected by the layout, assuming the screen size is fixed. The display size may affect the time factor to comprehend the data object's label (t_{view} and $t_{overview}$). The layout determines the number of data objects displayed simultaneously (n_{batch}), affecting the effectiveness of batch edit commands. For layouts with larger n_{batch}, issuing a batch edit command can edit the labels of more data objects. Section 5.4 discusses this factor.

- **Application scenario**: The user operation time costs may depend on the quality assurance process's application scenarios (e.g., the ones introduced in Section 5.2.1). For example, the complexity and time cost of comprehending the data labels (t_{view} and $t_{overview}$) may depend on the application scenario. Section 5.4 discusses this factor.

- **Availability of interface functions**: Section 5.3.1 introduces a general QA workflow that concerns several functions commonly but not always offered in QA interfaces, such as batch editing and the rank method. The availability of these functions may affect the time cost of the QA process. Section 5.5 discusses

the influence of introducing batch edit commands. Section 5.7 discusses the influence of introducing batch edit commands.

- **User's label strategy**: As the QA interface provides more functionalities, there may be multiple methods to accomplish the same goal. The user's strategy to select a method for accomplishing the goal affects the time cost. Section 5.5 discusses this factor.

- **Default label accuracy**: The accuracy of default labels affects the number of times the user needs to use single edit and batch edit commands. Section 5.6 discusses this factor.

- **Rank method**: The rank method may change the distribution of label categories in each batch. Rank methods are essentially instances of "data object selection" modules introduced in Section 4.3.4. Effective rank methods may alter the distribution to create more opportunities for batch edit commands to label multiple data objects with one click. Section 5.7 discusses this factor.

Note that for each type of user operation, we do not assume the time costs of the operations are the same as the original KLM assumes. We model user operation time costs as functions of the **layout** of the grid-based interface and the **application** that the dataset to be quality-assured comes from.

5.3.4 Scope and Assumptions

In the following sections, we set the following scope for our simulation-based evaluation:

- We focus on the scenario of **quality assurance of classification labels**. The following sections demonstrate the analysis of quality assurance for binary classification labels, with the two classes being "positive" and "negative". Our analysis can easily be extrapolated to multi-class classification scenarios.

- We focus on the **task completion time** as the evaluation metric.

We make the following assumptions to simplify the modeling and simulations:

- We assume **error-free execution** in the user operations. The user is assumed to be able to recognize the correct labels of the data objects and does not accidentally activate incorrect commands when using QA interfaces. Fully capturing the user strategies and potential mistakes is intractable due to the current insufficient understanding of the human decision process. Thus, a general strategy in model-based evaluation methods is to model what the users

should do instead of what the users will do [238]. We essentially give a lower-bound estimation by adopting the original assumption in the Keystroke-Level Model that the user makes no mistakes.

- We assume the **machine operations are fast** and their time costs are ignorable compared with user operation costs. Thus, we focus on the time cost of user operations but not machine operations.
- We assume the user operations we concern with are **sequential** instead of parallel. Assuming operations are sequential, $GetCost$ in Algorithm 3 is a simple summation of the multiplications of each operation's time cost and the number of executions. $GetCost(ops) = \sum_{op} N_{op} t_{op}$ where op denotes a type of user operations, N_{op} denotes the number of this type of user operation, and t_{op} denotes the time cost to execute the operation once.

The following sections introduce different technical aspects of our simulation model. We gradually bring out factors to be modeled, increasing the model's complexity.

5.4 The Factors of Interface Layout and Application

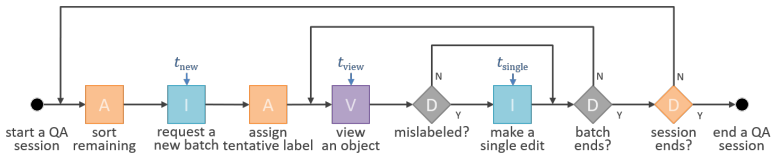


Figure 5.4: A workflow of human and machine operations in a grid-based interface without batch edit functions. This workflow simplifies the workflow in Fig. 5.3 by removing operations related to batch edit.

The following introduces simulations of task completion time with different interface layouts and application scenarios. We introduce a method to estimate user operations' time costs in the grid panel for editing labels. We show how simulation can help find an optimal layout for the grid-based interface.

This section considers a simplified QA interface template than the one in Section 5.2.2. This simplified interface template provides the new batch and single edit commands but does not provide batch edit commands. Fig. 5.4 shows its workflow.

5.4.1 Datasets to be Quality-Assured

In the simulation, we consider three real datasets to be quality-assured coming from the QA scenarios introduced in Section 5.2.1.

Visual Object Detection: This scenario concerns detecting visual objects from historical visualizations as introduced in Section 3.4.2 (see Fig. 5.1(b_1)). An ML model has assigned default labels to 3975 candidates of visual objects in John Snow's cholera map [250]. A post hoc analysis reveals that the model predictions have a normalized confusion matrix $cm = \begin{pmatrix} 86.98\% & 0.87\% \\ 0.53\% & 11.61\% \end{pmatrix}$, i.e., $\frac{n_{TN}}{n} = 86.98\%$, $\frac{n_{FP}}{n} = 0.87\%$, $\frac{n_{FN}}{n} = 0.53\%$, $\frac{n_{TP}}{n} = 11.61\%$. We refer to this dataset as *JS-block*.

Visual Object Grouping: This scenario concerns classifying pairs of visual objects to group visual objects as introduced in Section 3.4.2 (see Fig. 5.1(b_2)). An ML model that determines if a pair of visual objects should be grouped or not assigned default labels to 782 candidates of visual object pairs. A post hoc analysis reveals that the model predictions have a normalized confusion matrix $cm = \begin{pmatrix} 70.15\% & 0.12\% \\ 0.36\% & 29.37\% \end{pmatrix}$, i.e., $\frac{n_{TN}}{n} = 70.15\%$, $\frac{n_{FP}}{n} = 0.12\%$, $\frac{n_{FN}}{n} = 0.36\%$, $\frac{n_{TP}}{n} = 29.37\%$. We refer to this dataset as *JS-grouping*.

Solar Panel Detection: This application concerns classifying aerial images to detect solar panels (see Fig. 5.1(a)). This dataset is provided by Statistics Netherlands[1]. An ML model assigned default labels to 1278 aerial images. A post hoc analysis reveals that the model predictions have a normalized confusion matrix $cm = \begin{pmatrix} 64.82\% & 5.60\% \\ 5.44\% & 24.14\% \end{pmatrix}$, i.e., $\frac{n_{TN}}{n} = 64.82\%$, $\frac{n_{FP}}{n} = 5.60\%$, $\frac{n_{FN}}{n} = 5.44\%$, $\frac{n_{TP}}{n} = 24.14\%$. We refer to this dataset as *SP-image*.

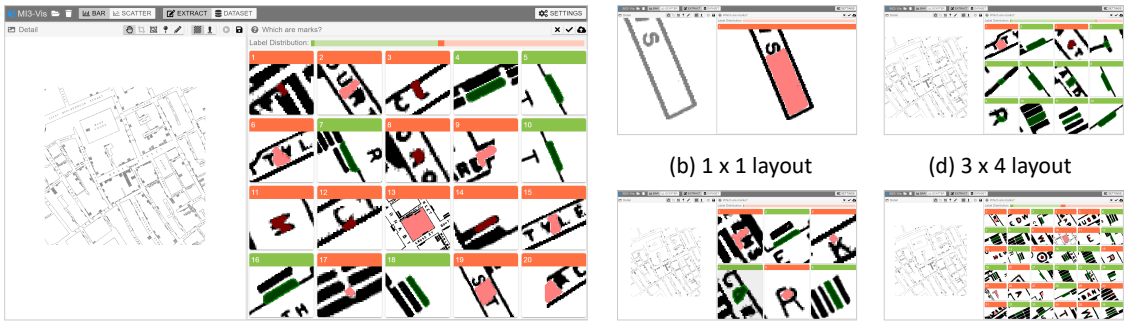


Figure 5.5: Different layouts of a grid-based interface for visual object detection in JS-block dataset: (a) 4 × 5 layout, (b) 1 × 1 layout, (c) 2 × 3 layout, (d) 3 × 4 layout, and (e) 6 × 6 layout.

5.4.2 Conceptual Model for Simulation

Fig. 5.4 shows a QA workflow. Consider a grid panel with K grid cells. The grid panel provides two commands for the user to operate: **request a new batch** and **make a single edit**. We model the total time for a round of quality assurance operations for K grid cells as:

$$T_{round} = t_{new} + K \cdot t_{view} + (K_{FP} + K_{FN}) \cdot t_{single} \tag{5.1}$$

[1] https://www.cbs.nl/

K_{FP} and K_{FN} are the numbers of false positive and false negative labels in the batch, which are known values. We derive this linear model based on the assumption that the user operations are sequential instead of parallel, and thus the total time cost is a summation of the user operation time costs. For a session with in total n data objects, denote its overall confusion matrix as $\left(\begin{smallmatrix} n_{TN} & n_{FP} \\ n_{FN} & n_{TP} \end{smallmatrix}\right)$. One can extrapolate the time cost of a round and deduce the total time cost of a session: $T_{session} = \lceil \frac{n}{K} \rceil t_{new} + n \cdot t_{view} + (n_{FP} + n_{FN}) \cdot t_{single}$. As long as we have the operator time costs t_{new}, t_{view}, and t_{single}, we can estimate the time T_{round} and $T_{session}$.

5.4.3 Estimating Operator Time Costs

There are various approaches to the estimation of operator time costs t_{new}, t_{view}, and t_{single}. One may recruit potential users to use a QA interface and use eye-tracking, video recording, or interaction logs to capture detailed timing data for different layout designs. However, such processes usually require a relatively complex experimental setup unavailable or inconvenient to most software development projects. In the model-based evaluation literature, another common practice is to reuse time costs estimated by previous work [48]. To accomplish a swift evaluation, one may also plug in hypothetical values for the operator time costs, such as assuming all types of operations are of the same cost, as we practiced in Chapter 3. While these approaches are all feasible for the scenario of QA simulation, in the following, we introduce an approach to get a realistic estimation of operator time costs swiftly.

Method

We sample T_{round} values through self-experimentation under the conditions of different values of n_{batch} and $K_{FP} + K_{FN}$ and different layout designs. We then use numerical analysis and modeling to obtain t_{new}, t_{view}, and t_{single} values. This method allows QA interface designers to:

- gain a deeper understanding of viewing actions and interface operations in a QA application,
- use their knowledge and intelligence to address anomalies and artifacts due to sparse sampling, and
- use the obtained models to inform conditions (e.g., layout designs) that have not been experimented with.

Appendix C details the process for obtaining the numerical models of t_{new}, t_{view}, and t_{single} for the three usage scenarios in Section 5.4.1. We obtain the experiment records through self-experimentation by one of the authors. Here we outline the process to generate operator time cost estimations in Fig. 5.6.

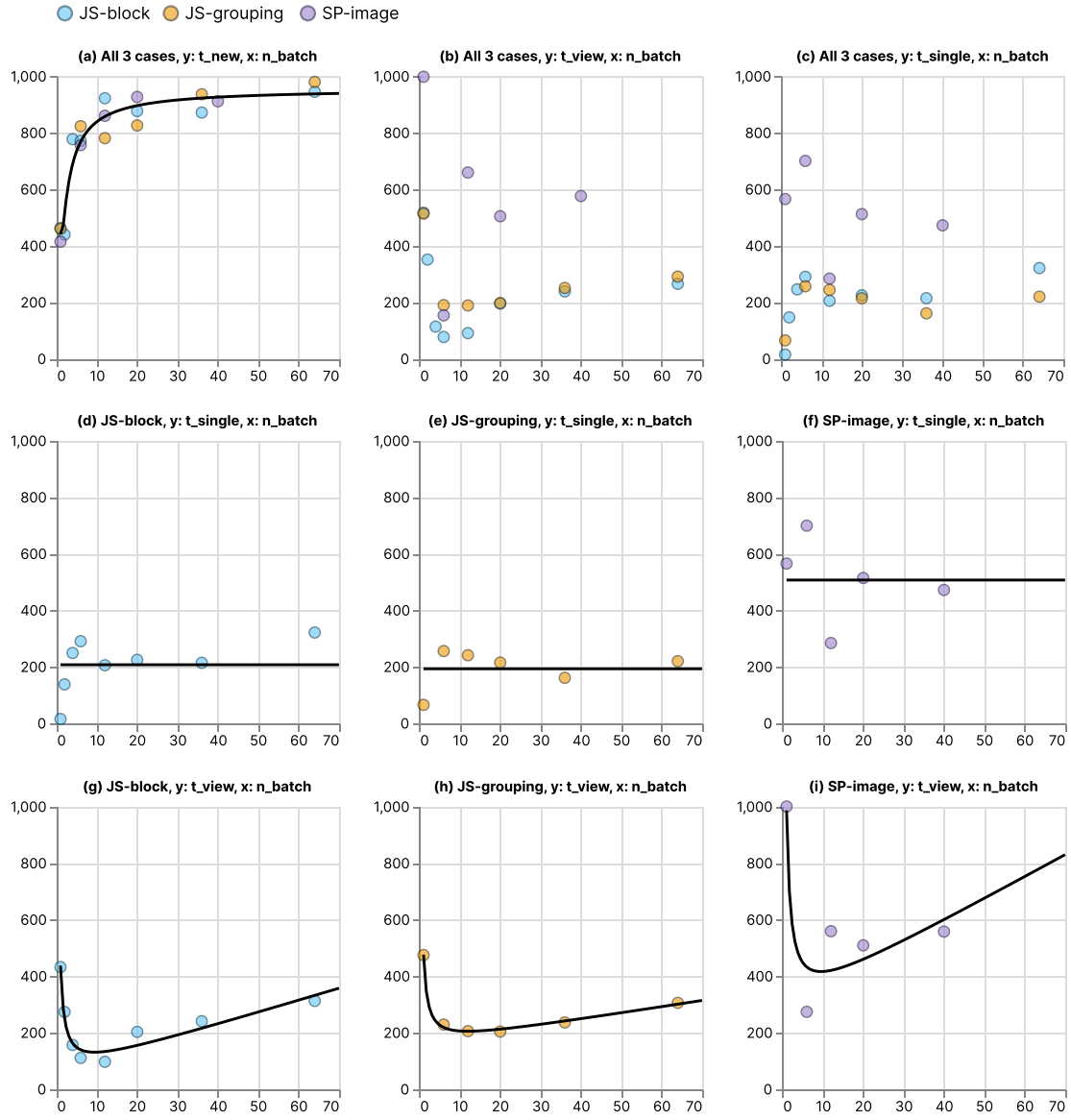


Figure 5.6: Summary of the experiment result and fitted models: (a) Estimation of t_{new} by multiple linear regression and model of t_{new}. The selected model is $y = a + b/x + c/x^2$. (b) Initial estimations of t_{view} by multiple linear regression. (c) Initial estimations of t_{single} by multiple linear regression. (d - f) Models of t_{single} for the 3 cases. The selected model is $y = a$. The data points are t_{single} reestimated by putting t_{new} back. (g - i) Models of t_{view} for the 3 cases. The selected model is $y = a + bx + c/x$. The data points are t_{view} reestimated by putting t_{new} and t_{single} back. The x-axes of the figures are the number of grid cells n_{batch}. The y-axes of the figures are the corresponding operator time costs t_{new}, t_{view}, or t_{single}. The unit of all the time costs is milliseconds.

Procedure and Results

The values of operator time costs t_{new}, t_{view}, and t_{single} may depend on the **layout** of the QA interface and the **application**. For example, the time of viewing each grid cell, i.e., t_{view}, may vary for different sizes of grid cells in the editing panel (dependent on layout) and the complexity of the recognizing data objects shown in these grid cells (dependent on application). Therefore, we estimate operator time costs separately for different layouts and applications in the following.

For each application scenario introduced in Section 5.4.1, we create several prototype QA interfaces. The interfaces are like the ones in Fig. 5.2 but have different layout designs. Fig. 5.5(b - e) show a few grid layout design options in the QA interface for the JS-block dataset. We sample eight grid layouts for quality-assuring the ML model for JS-block (i.e., 1×1, 1×2, 2×2, 2×3, 3×4, 4×5, 6×6, and 8×8), six layouts for JS-grouping (i.e., 1×1, 2×3, 3×4, 4×5, 6×6, and 8×8), and five layouts for SP-image (i.e., 1×1, 2×3, 3×4, 4×5, and 5×8). For each grid layout, we take a minimum of seven samples of T_{round} with different values $(K, K_{FP} + K_{FN})$ (see Table C.1 and Table C.2).

We assume the total time cost follows the linear model in Equation 5.1. Thus, for each combination of layout and application, given the observations with different $(K, K_{FP} + K_{FN})$, we use multiple linear regression to estimate t_{new}, t_{view}, and t_{single} values. The estimation results are shown in Fig. 5.6(a - c) and numerically in Table C.3. Note that the estimations of the user operation time costs are mostly statistically significant with high R^2, which suggests that our linear model in Equation 5.1 is numerically accurate. Thus, our assumption that the user operations are always sequential instead of parallel is realistic.

Then, for each application, we fit a function of the operator time costs regarding layout (parameterized as n_{batch}). Our purpose in fitting the function is not to obtain an ultimate model of the time costs. Using the fitted functions, we can obtain smoothed estimations of the operator time costs and predict the operator time costs for layouts we have not experimented with.

The t_{new} values for three usage scenarios are shown in three different colors (in Fig. 5.6(a)). They show similar trends concerning the number of grid cells. Based on this observation, as well as the rationale that the time required for issuing a new batch command is likely similar among the three usage scenarios, we decide to fit a single model to the three sets of data points.

Details on Curve Fitting

Firstly, we use a set of elementary functions (e.g., constant, polynomial, exponential, logarithm) to fit the operator time costs as a function of n_{batch}. Then, we make several attempts to formulate a semantic explanation of those functions with low error measures and abandon the functions for which we cannot find an intuitive

explanation. We choose a function as the final model based on fitting errors and interpretability. Fitting the functions enables us to smooth the estimations of the operator time costs. In the following, we introduce the process of curve fitting. More details can be found in Appendix C.

We use n_{batch} to be the parameterization of the layout. One may expect that for different layouts (e.g., 2×3 and 3×2) with the same n_{batch}, the operator time costs may still be different. While it is possible to fit functions with the number of rows and the number of columns being the functions' inputs, we use n_{batch} because the ratio of the number of rows and number of columns in all the layouts we have experimented are close and are all between 1 : 1 and 1 : 2.

Fitting t_{new}: For the initial estimations of t_{new} in Fig. 5.6(a), formulae $a + b/x$, $a + b/x + c/x^2$, and $a + b/x + c/x^2 + d/x^3$ are found by numerical analysis to be among the best candidate models for 2-, 3-, and 4-parameter models respectively. With a positive a and a negative b, these formulae suggest that there may be a maximum time cost in activating a new batch, while the smaller grid cells can reduce the cost, possibly because of the easiness of deciding if a QA round is completed. The curve shown in Fig. 5.6(a) is: $\mathbf{t_{new} = 958.0085 - \frac{1254.8737}{n_{batch}} + \frac{739.4750}{n_{batch}^2}}$

As shown in Fig. 5.6(b), the t_{view} values estimated based on Eq. 5.1 exhibit different patterns for the three usage scenarios. This is understandable, as viewing and reasoning about the three types of data objects may incur different time costs. In particular, viewing solar panel images is, in general, more time-consuming, except that the second data point seems to be an outlier. Meanwhile, as shown in Fig. 5.6(c), the estimated t_{single} values for the first two usage scenarios (i.e., JS-block and JS-grouping) are reasonably similar, while the t_{single} values for the solar panel usage scenario are in general higher than those of the other two usage scenarios.

We thus treat t_{view} and t_{single} for the three usage scenarios separately. We choose to model t_{single} next because the time cost for issuing a single edit command (which is mostly about human motor control) is expected to be less affected by the size of the data objects being displayed than the time cost of viewing these data objects (which requires comprehension). Moreover, there are fewer single edit actions than viewing actions, and modeling t_{single} is expected to affect t_{view} less than modeling t_{view} before t_{single}.

Fitting t_{single}: Before modeling t_{single}, we reestimated the t_{single} data points by replacing the t_{new} variable in Eq. 5.1 with modeled t_{new} data values. The reestimated t_{single} data points for the three usage scenarios are shown in Fig. 5.6(d - f) respectively. Following the same procedure for gathering and analyzing candidate models, we have selected a constant model for each usage scenario. **The three constants are 208.0818, 193.8312, and 507.9517 for JS-block, JS-grouping, and SP-image, respectively.**

Fitting t_{view}: In the third iteration, we first reestimate the t_{view} data points using the modeled t_{new} and t_{single} data. As shown in Fig. 5.6(g - i), the modeling of t_{new}

and t_{single} hardly caused a noticeable displacement of the t_{view} data points. We can also easily observe a common phenomenon that the t_{view} values are higher when there are too many or too few grid cells (equivalently, when the sizes of grid cells are too small or too large). Following the same procedure for gathering and analyzing candidate models, we select the formula $y = a + bx + c/x$ for all three usage scenarios as it semantically captures the effect of grid sizes with its second and third terms. The three models are:

- **for JS-block:** $\mathbf{t_{view} = 50.4283 + 4.3202 n_{batch} + \frac{383.5033}{n_{batch}}}$
- **for JS-grouping:** $\mathbf{t_{view} = 151.9951 + 2.2618 n_{batch} + \frac{322.6178}{n_{batch}}}$
- **for SP-image:** $\mathbf{t_{view} = 267.0109 + 7.9103 n_{batch} + \frac{712.8504}{n_{batch}}}$

Following the curve fitting and reestimation steps, we obtain the operator time costs as shown in Table C.4.

Findings

Through the operator time cost estimations, we have the following observations:

- The initial estimations of the operator time costs exhibit low square errors as shown in Table C.3. The estimation is conducted with multiple linear regression because we assume the user operations are sequential and the total time cost is a summation of the operator time costs. The low square errors imply that the assumption is reliable, at least for numeric modeling purposes.
- The operator time costs depend on layout and application, as shown in Fig. 5.6. It implies that QA interface designers need to optimize the layout to improve users' productivity.
- $\mathbf{t_{view}}$ follows a U-shaped trend with the number of grid cells in the grid layout (as shown in Fig. 5.6(g - i)). This implies that grid cells that are too large or too small are hard to view and thus cost more time. For small grid cells, the user may need to inspect them more carefully, which costs more time. For large grid cells, we interpret that the user may need additional pupil movements for the foveal vision to cover the visual representation of the data object.
- The estimations of $\mathbf{t_{view}}$ in SP-image are much larger than those in JS-block and JS-grouping. This implies that it is harder for the user to do quality assurance in the SP-image application than in the JS-block and JS-grouping applications. We conceive it is harder for the user to comprehend the aerial images in SP-image than the visual objects in JS-block and JS-grouping.

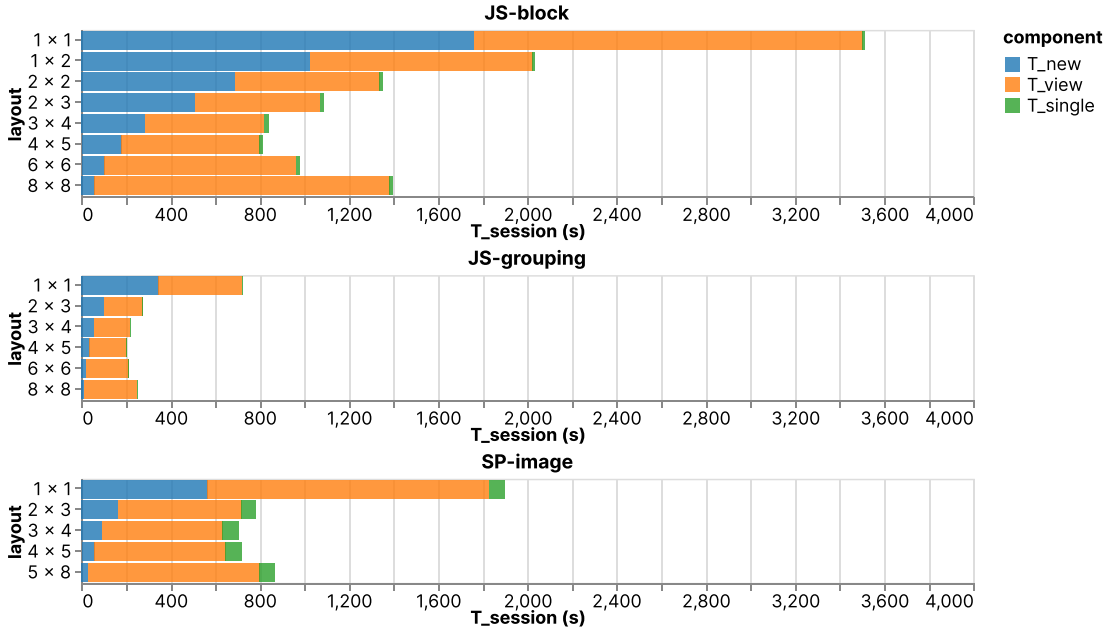


Figure 5.7: Simulation of layouts: The effect of grid layout in the three applications. For all the applications, the total time cost follows a U-shaped curve. For JS-block, the minimal is achieved at 4×5. For JS-grouping, the minimal is achieved at 4×5. For SP-image, the minimal is achieved at 3×4.

Algorithm 4 Grid-Based Interface (Single Edit)

Input: true labels Y^*, default labels Y
Output: user operation sequence $ops = (op_1, ..., op_m)$

$i \leftarrow 1$
$ops \leftarrow []$
do
    ops.push("new") ▷ request a new batch
    $batchEnd \leftarrow min(i + n_{batch}, n)$
    for $j \in (i, i+1, ..., batchEnd)$ **do**
        ops.push("view") ▷ view an object
        if $Y[j] \neq Y^*[j]$ **then** ▷ mislabeled?
            ops.push("single") ▷ make a single edit
        end if
    end for
    $i = i + n_{batch}$
while $i \leq n$ ▷ session ends?
return ops

5.4.4 Simulations and Observations

In the following, we use simulation to examine the influence of application and layout on the total time cost $T_{session}$. We demonstrate using simulation to find the optimal layout of QA interfaces.

Simulation Setup

We conduct simulations to estimate $T_{session}$ with regards to different applications and layouts following the workflow in Fig. 5.4. Algorithm 4 describes how to simulate an operation sequence in a simulation trial.

We run the simulations with the following parameter values:

- interface layout: all the $layout$ for which we have estimated operator time costs as discussed in Section 5.4.3.
- dataset parameters: 3 conditions corresponding to the dataset size n and normalized confusion matrix cm of the three applications JS-block, JS-grouping, and SP-image introduced in Section 5.4.1.
- operator time costs: t_{new}, t_{view}, and t_{single} to be the values measured for the given $application$ and $layout$ as shown in Fig. 5.6 and Table C.4.

Result and Analysis

Fig. 5.7 shows the simulation results in relation to different grid layouts and applications. The bar charts in the three applications all exhibit a U-shape pattern. **For JS-block, the minimal total time cost for a QA session is achieved with the** 4×5 **layout. For JS-grouping, the minimum is also with the** 4×5 **layout. For SP-image, the minimum is with the** 3×4 **layout.** In Section 3.4.4, we initially designed an interface with 3×2 layout as shown in Fig. 3.11. The simulation result informs us that for the consideration of user time costs, it is better to switch our layout to 4×5.

The default labels for datasets to be quality-assured have reasonably high accuracies (1.4% for JS-block, 0.48% for JS-grouping, and 11.04% for SP-image). Thus, the total cost of single edit operations (T_{single}) is low compared with the total cost of the operations for starting a new batch (T_{new}) and the total cost of the viewing actions (T_{view}). The optimal layout reflects a trade-off between T_{new} and T_{view}.

Through the simulation, we observe that layout has a significant impact on $T_{session}$. The optimal layout is dependent on the QA application. **There is no single layout that is optimal for all the application scenarios.** A design implication is that for QA interfaces, UI specialists need to optimize the layout with regard to the concerned application. Alternatively, QA interfaces may provide layout switching functions for the user.

5.5 The Factor of User Label Strategy

Section 5.4 has introduced the setting where using "single edit" commands is the only way to edit labels. For the grid-based interface in QA, except for the 1×1 layout, all other layouts present multiple data objects in a batch. Thus, a natural extension of the interface functionality is to enrich the label editing methods by providing batch edit commands that simultaneously edit the label of multiple data objects.

For binary classification, imagine a QA scenario in which among 20 data objects in a batch, 10 are TP, 8 are FN, and 2 are FP. If there are no batch edit commands in the interface, a user needs to issue 10 single edit commands to correct the labels. If there is a batch edit command called "all positive" that sets all the data objects' label category to be "positive" in the interface, the user can achieve the same goal with three commands, i.e., "all positive" first, the two "single edit" commands for the two false-positive labels.

Users will likely use their intelligence to take such a shortcut when batch edit commands are available. This section uses simulation to examine whether providing such batch edit commands is beneficial. When a QA interface provides batch edit commands, the user has multiple methods to accomplish a QA session, such as sticking to using only the single edit commands or combining single edit commands and batch edit commands. In the following, we also model the user's label strategy.

As the application scenarios of JS-block and JS-grouping concern active learning workflows, as illustrated in Fig. 5.1(b), a user can dynamically decide when to stop the active learning after labeling k data objects interactively. If k is small, the cost of manual labeling during active learning is low, but the ML model is likely less accurate and more erroneous labels to be correct during QA. On the other hand, the less accurate ML results are, the more benefit the batch edit commands can bring. Hence, it is interesting to simulate the impact of a user's strategies for using batch edit commands in conjunction with different accurate levels of the ML results.

5.5.1 Conceptual Model for Simulation

After introducing the batch edit commands, the user has multiple methods to carry out the QA processes. For example, the user may stick to using the single edit command as is in Section 5.4, or the user may decide to combine batch edit commands and the single edit commands. Moreover, the user may not use the same method for all the batches. The user may decide to apply different methods depending on the distribution of TP, TN, FP, and TN in each batch.

In this section, we model the user's strategy of choosing among the methods. The user strategy corresponds to the concept of "selection rules" in GOMS models [133].

Batch Edit Methods

Fig. 5.3 extends the basic QA process in Fig. 5.4 with batch editing commands. For quality assurance in binary classification, we consider three batch edit commands:

- "All positive": label the whole batch of data objects "positive".
- "All negative": label the whole batch of data objects "negative".
- "Inverse all": label the whole batch of data objects to the logical inverse of their current labels.

If a user does not make any mistakes in observation and command selection, each batch edit command should be selected no more than once. When the three batch edit commands are available, a user can observe all data objects in the editing panel and choose one of the following four methods to finish a round of quality assurance:

- **Baseline (B)**: using only "single edit" commands to correct all negative positive/negative errors (number of single edit $N_{single} = K_{FP} + K_{FN}$).
- **Mostly positive (P)**: issuing an "all positive" command, and using "single edit" commands to correct remaining errors ($N_{single} = K_{FP} + K_{TN}$).
- **Mostly negative (N)**: issuing an "all negative" command, and using "single edit" commands to correct remaining errors ($N_{single} = K_{FN} + K_{TP}$).
- **Mostly wrong (W)**: issuing an "inverse all" command, and using "single edit" commands to correct remaining errors ($N_{single} = K_{TP} + K_{TN}$).

The set of possible label methods depends on the available commands (denoted as $cmds$) for editing labels. When not all four label commands are provided in the interface, the corresponding label method (B, P, N, or W) is not available.

Typically, "single edit" commands are provided in QA interfaces to guarantee that all the data objects' labels can be edited individually. Thus we always assume $SingleEdit \in cmds$, and we refer to the label method of always using single edit as the baseline label method. Let $cmds_{single} = \{SingleEdit\}$ denote the label edit commands available for a QA interface without batch edit command. When "single edit" commands and the three "batch edit" commands are all provided in the interface, $cmds = \{SingleEdit, AllPositive, AllNegative, InverseAll\}$. We denote this value with $cmds_{batch}$.

The time costs of a round of quality assurance T_{round} for these four methods are:

B: $T_{round} = T_0 + (K_{FP} + K_{FN}) \cdot t_{single}$

P: $T_{round} = T_0 + t_{allPositive} + (K_{FP} + K_{TN}) \cdot t_{single}$

N: $T_{round} = T_0 + t_{allNegative} + (K_{FN} + K_{TP}) \cdot t_{single}$

W: $T_{round} = T_0 + t_{inverseAll} + (K_{TP} + K_{TN}) \cdot t_{single}$

where $T_0 = t_{new} + t_{overview} + K \cdot t_{view}$ and $t_{overview}$, $t_{allPositive}$, $t_{allNegative}$, and $t_{inverseAll}$ are the time costs for overviewing all grid cells in the editing area and issuing one of the three batch edit commands, respectively. Which method is the most efficient is dependent on the confusion matrix $\left(\begin{smallmatrix} K_{TN} & K_{FP} \\ K_{FN} & K_{TP} \end{smallmatrix}\right)$ of the batch of data objects.

Parameterizing User Label Strategy

The effectiveness of using the label edit commands provided in $cmds$ depends on a user's ability to choose the right method (that minimizes the time cost) when facing a new batch of data objects. We parameterize this factor of a user's ability to make method choices so that we can simulate user strategies with different parameter values. The parameterization of user strategies corresponds to an implementation of "SelectEditCommand" in Algorithm 3. We introduce two model parameters:

- **User skill level** $u_{bcsl} \in [-1, 1]$ represents different levels of user skills, with 1 being the best and -1 being the worst.
- **User strategy uncertainty** $u_{bcsu} \in [0, 1]$ represents randomness in choosing among the methods.

The overall idea of the parameterization is that each label method is associated with a time cost and can be ranked by the time cost. A higher skill level corresponds to choosing a smaller rank index. The choice of a rank index may be associated with an error range, and the strategy uncertainty captures the width of the error range.

If a user does not use batch edit commands and always chooses the baseline method **B**, we have $u_{bcsl} = 0$. When u_{bcsl} is set to 1, the simulation model considers that the user is highly experienced and will always choose the best method for each batch. Let N_m be the total number of methods. $N_m = 4$ when the batch edit commands are all available. $N_m = 1$ when only single edit commands are available. A simulation model knows the ground truth labels. Therefore, it can sort all label methods by time costs from 0 to $N_m - 1$ with 0^{th} being the best. The strategy of always choosing the method at the 0^{th} position corresponds to $u_{bcsl} = 1$. The strategy of always choosing the method at the $(N_m - 1)^{th}$ position corresponds to $u_{bcsl} = -1$.

When $0 < u_{bcsl} < 1$, the simulation model chooses the label method at the k-th position between the best method at position 0 and the baseline method at position bl, such that $k = \text{round}\big((1 - u_{bcsl}) \cdot bl\big)$. If the baseline method is the best, $bl = 0$ and $k = 0$.

When $-1 < u_{bcsl} < 0$, the simulation model chooses the k^{th} label method between the baseline method at position bl and the worst at position $(N_m - 1)$, such that

$k = \text{round}\big(-u_{bcsl} \cdot (N_m - 1 - bl) + bl\big)$. Usually, users will not deliberately choose a batch edit command that would lead to more user operations than the baseline method. $u_{bcsl} < 0$ is thus rare.

When $u_{bcsu} = 0$, the selection of the label method is absolutely certain and is based on u_{bcsl} only. When $u_{bcsu} = 1$, the selection is absolutely uncertain and is randomly chosen among all N_m strategies. When $0 < u_{bcsu} < 1$, the simulation model randomly chooses a label method ranked between the a-th and b-th positions, such that $a = \max(0,\ k - \delta)$ and $b = \min(N_m - 1,\ k + \delta)$, where $\delta = \text{round}(u_{bcsu} \cdot N_m)$.

5.5.2 Estimating Operator Time Costs

In principle, $t_{overview}$, $t_{allPositive}$, $t_{allNegative}$, and $t_{inverseAll}$ can be estimated in a manner similar to that for t_{new}, t_{view}, and t_{single} in Section 5.4. In the following, we consider a discount process for estimating these operator time costs.

It is intuitive to anticipate that $t_{allPositive}$, $t_{allNegative}$, and $t_{inverseAll}$ are close to t_{new} because their corresponding buttons are all located at the header toolbar. We thus estimate $t_{allPositive} = t_{allNegative} = t_{inverseAll} = t_{new}$ based on this assumption. Because the frequency of using commands "all positive", "all negative", and "inverse all" is much lower than other operations, a minor discrepancy in estimating $t_{allPositive}$, $t_{allNegative}$, and $t_{inverseAll}$ has limited impact.

The time that a user spends for overviewing all data objects ($t_{overview}$) before issuing (or skipping) a batch edit command can be costly when n_{batch} is large. It likely increases with n_{batch} as the user needs to comprehend more data objects. Meanwhile, there may be a small reduction in the time used for examining an individual data object (t_{view}) after a batch edit command is issued (or skipped), taking advantage of the previous observation from the overview. To capture these intuitions, in the following simulation, we assume a simple linear model $t_{overview} = n_{batch} \times 0.025$ with the unit being second to ensure $t_{overview}$ is monotonous with n_{batch}. We discount t_{view} by 0.0125 seconds when an overview is conducted.

5.5.3 Simulations and Observations

Through this set of simulations, we aim to identify the influence of introducing batch edit commands and user's label strategies to $T_{session}$ and to the optimal point of $layout$.

Simulation Setup

We run the simulations with the following parameter values:

- interface layout: 7 conditions of $layout$ being $\{1 \times 2, 2 \times 2, 2 \times 3, 3 \times 4, 4 \times 5, 6 \times 6, 8 \times 8\}$.

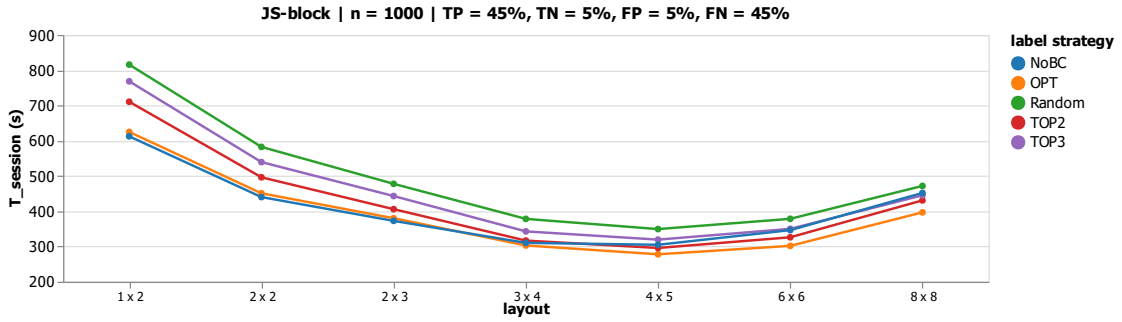


Figure 5.8: Variations of label strategy: The simulation result of total time cost $T_{session}$ with 5 different label strategies: NoBC, OPT, Random, TOP2, TOP3. The starting point of the vertical axis is 200 instead of 0 to highlight the differences in vertical positions.

- label strategy: 5 conditions of $(cmds, u_{bcsl}, u_{bcsu})$ corresponding to different label strategies by the user
  - **NoBC**: never uses batch edit commands ($cmds = cmds_{single}$, $u_{bcsl} = 0$, $u_{bcsu} = 1$).
  - **OPT**: always choose optimal combinations of batch and single edit commands ($cmds = cmds_{batch}$, $u_{bcsl} = 1$, $u_{bcsu} = 0$).
  - **Random**: choose a random combination of batch and single edit commands ($cmds = cmds_{batch}$, $u_{bcsl} = 1$, $u_{bcsu} = 1$).
  - **TOP2**: choose (randomly) the top 2 combinations of batch and single edit commands ($cmds = cmds_{batch}$, $u_{bcsl} = 1$, $u_{bcsu} = 0.25$).
  - **TOP3**: choose (randomly) the top 3 combinations of batch and single edit commands ($cmds = cmds_{batch}$, $u_{bcsl} = 1$, $u_{bcsu} = 0.5$).
- dataset parameters: $n = 1000$ and $cm = \begin{pmatrix} 0.05 & 0.05 \\ 0.45 & 0.45 \end{pmatrix}$.
- operator time costs: t_{new}, t_{view}, $t_{overview}$, t_{single}, and t_{batch} to be the values measured for the JS-block application for any given $layout$.

We run ten repeated simulation trials with different random seeds for each combination of simulation parameters. The random seed determines the random permutation of data objects.

Result and Analysis

Fig. 5.8 shows the simulation result. The following summarizes our observations.

Introducing batch edit commands saves user effort: It can be seen that for all the layouts with more data objects than 3×4, the label strategy OPT (choosing the optimal combination of batch and single edit commands in a QA interface with batch

edit commands) is consistently better than NoBC (using a QA interface without batch edit commands). The minimal $T_{session}$ is achieved by OPT at 4×5 layout.

User's label strategy influences the time cost: If we ignore NoBC, which is not combining batch edit commands, for all the other 4 label strategy configurations with the same skill level u_{bcsl}, a lower skill uncertainty u_{bcsu} reduces the session time cost.

Similar to Fig. 5.7, the time cost follows a U-shaped curve. With the increase of n_{batch} of the layout, NoBC's relative performance, compared with the other 4 strategies, is consistently decreasing. This shows that for larger n_{batch}, the benefit of using batch edit commands is larger.

5.6 The Factor of Default Label Accuracy

QA workflows require the user to verify and correct machine predictions. While it is intuitive that accurate machine predictions save user effort, it is not straightforward to what extent accurate predictions reduce task completion time. This section investigates the factor of default label accuracy to the QA time costs.

In practice, UI specialists may conduct such simulations to examine how much time can be saved by improving machine predictions. In this way, UI specialists can decide whether it is worthwhile to work on improving the default label accuracy or it is more cost-effective to work on improving other aspects of the QA interface.

5.6.1 Simulations and Observations

Through this set of simulations, we examine the influence of default label accuracy to $T_{session}$, and to the optimal point of $layout$.

Simulation Setup

We run the simulations with the following parameter values:

- interface layout: 5 conditions of $layout$ being $\{1 \times 2, 2 \times 2, 2 \times 3, 3 \times 4, 4 \times 5, 5 \times 6, 8 \times 8\}$.
- dataset parameters: $n = 1000$ and 5 condition of cm being:

  $$\{\begin{pmatrix} 0.5a & 0.5-0.5a \\ 0.5-0.5a & 0.5a \end{pmatrix} | a \in \{0.5, 0.6, 0.7, 0.8, 0.9\}\}$$

  The 5 conditions of cm correspond to different default label accuracies: 0.5, 0.6, 0.7, 0.8, 0.9.
- operator time costs: t_{new}, t_{view}, $t_{overview}$, t_{single}, and t_{batch} to be the values measured for the JS-block application for any given $layout$.

- label strategy: $cmds = cmds_{batch}$, $u_{bcsl} = 1$, and $u_{bcsu} = 0$. With this configuration, the optimal label strategy is used for every batch.

Ten repeated simulation trials with different random seeds are run for each combination of simulation parameters.

Result and Analysis

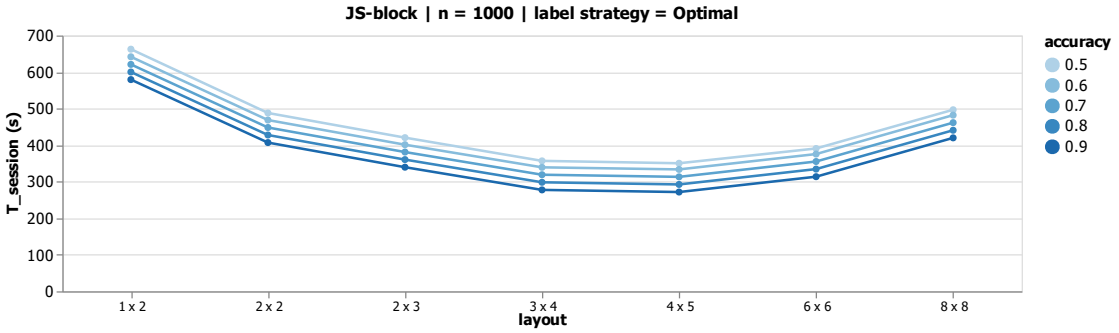


Figure 5.9: Variations of default label accuracy: The simulation result of total time cost $T_{session}$ with 5 conditions of default label accuracy, including 0.5, 0.6, 0.7, 0.8, and 0.9.

Fig. 5.9 shows the simulation result. It can be seen that for all the layouts, with the increase of default label accuracy, the session time cost consistently decreases, which is as expected. Meanwhile, although accurate models save much time, using a good model (accuracy = 0.9) with a bad layout (1 × 2) may cost more time than using a bad model (accuracy = 0.5) with a good layout (3 × 4). This observation implies that in QA interfaces, **good interface designs and algorithm designs are both important for saving user effort**.

5.7 The Factor of Rank Method

A general QA workflow involves rank methods to reorder data objects, as shown in Fig. 5.3. Rank methods can alter the local distributions of label categories and default labels' correctness in the batches.

Rank methods reduce the time cost by unbalancing the local confusion matrix $[K_{TP}, K_{TN}, K_{FP}, K_{FN}]$ of each batch. The effectiveness of rank methods in altering the distribution of label categories may affect the opportunities of using batch edit commands and influence the QA time cost.

To simulate rank methods, one may use actual implementations of the rank methods. For example, Zhang et al. [296, 297] simulate the number of user operations in active learning workflows. Their simulations use the actual implementations of scoring functions in active learning to rank the priority of data objects. Meanwhile, using actual

implementations in the simulations is inefficient, as each rank method needs to be implemented by the evaluator. In the following, we introduce a parameterization of rank methods. Using parameterization, we can simulate rank methods by configuring different parameter values. The parameterization approach also gives us more control over the characteristics of the rank methods in simulations.

5.7.1 Conceptual Model for Simulation

To capture the rank methods in the simulation model, it is desirable to model the "ability" of reordering algorithms to change the label category distribution in each batch. Modeling of the rank method corresponds to an implementation of "Rank" in Algorithm 3. In the following, we focus on the type of rank methods that can be modeled as selecting a subset of data objects according to some criteria and moving them to the front of the list of data objects. We refer to these rank methods as bipartition ("BiPart") rank methods.

Parameterizing Rank Method

We parameterize a rank method with a matrix $rm \in [0,1]_{c\times c}$ where c is the number of classes. $rm_{i,j}$ denote the rate of data objects with true label i and default label j that are moved to the front of the list by the rank method among all such data objects. For a binary classification problem, $rm = \left(\begin{smallmatrix} p_{TN} & p_{FP} \\ p_{FN} & p_{TP} \end{smallmatrix}\right)$. p_{TN} denotes that among all the n_{TN} true negatives in the dataset, $n_{TN} \times p_{TN}$ are moved to the front of the list. We refer to rm as the rank method's "selection matrix". The total number of objects selected by the rank method rm and moved to the front of the list is thus: $n_s = \sum_{i,j} rm_{i,j} \times n \times cm_{i,j}$. All the selected data objects are regarded as unordered. Thus, the selected data objects are randomly shuffled in the simulation. In the case of binary classification, $n_s = n_{TP}p_{TP} + n_{TN}p_{TN} + n_{FP}p_{FP} + n_{FN}p_{FN}$.

Examples

Consider a rank method that selects data objects with ground truth labels positive (i.e., true positives and false negatives) to the front of the list, with $rm = \left(\begin{smallmatrix} 0 & 0 \\ 1 & 1 \end{smallmatrix}\right)$. If a batch contains only a subset of these selected data objects, the user can use a batch edit command to set all their label categories to be "Positive".

Consider another rank method with $rm = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, i.e., selecting all data objects belonging to FN and FP. Such a rank method moves all the data object with incorrect default labels to the front, and all the user needs to do is use a batch edit command to inverse all of their labels.

In these two examples, the rank methods are good in that they create opportunities for the user to use the batch edit command effectively. The "ability" of a rank method is to make the probability distribution of TP, TN, FP, and FN labels as less uniform

as possible. Any high concentration of one group of TP, TN, FP, or FN and combined groups of (TP, FN), (TN, FP), (TP, TN), or (FP, FN) can be beneficial. What is undesirable is any reordering result that makes a batch of data objects featuring similar numbers of TP and FP labels or similar numbers of TN and FN labels. These are situations where batch edit commands are beneficial.

Quantifying Rank Method Performance

It is unrealistic to expect the rank method to be good enough to get all the TP/TN/FP/FN ranked top, as in the examples above. We quantify the goodness of a rank method by its difference from the ideal case. In general, we denote an ideal selection with a binary matrix: $target \in \{0,1\}_{c\times c}$ where $target_{i,j} = 1$ when data objects with true label i and default label j are target data objects that are intended to be selected.

As introduced above, a rank method rm selects $n_s = \sum_{i,j} rm_{i,j} \times n \times cm_{i,j}$ and moves them to the front of the list. As specified by the target matrix $target$, among all the selected data objects, $\sum_{i,j} rm_{i,j} \times n \times cm_{i,j} \times target_{i,j}$ of them are indeed the targets. Thus, the rank method precision is $\frac{\sum_{i,j} rm_{i,j} \times cm_{i,j} \times target_{i,j}}{\sum_{i,j} rm_{i,j} \times cm_{i,j}}$. The precision value can be used to quantify rank methods' performance. The precision depends on the choice of the target matrix.

5.7.2 Simulations and Observations

Through this set of simulations, we examine the influence of introducing rank methods to $T_{session}$. We expect good rank methods to reduce the time cost.

Simulation Setup

We run the simulations with the following parameter values:

- interface layout: 5 conditions of $layout$ being $\{2\times 2, 2\times 3, 3\times 4, 4\times 5, 6\times 6, 8\times 8\}$.
- rank method: 5 conditions of rm being

  $$\{\begin{pmatrix} \frac{1-a}{3} & \frac{1-a}{3} \\ a & \frac{1-a}{3} \end{pmatrix} | a \in \{0, 0.25, 0.5, 0.75, 1\}\}$$

  We have also simulated a scenario where no rank method is used and the dataset is randomly ordered.
- label strategy parameters: $cmds = cmds_{batch}$, $u_{bcsl} = 1$, and $u_{bcsu} = 0$. With this configuration, the optimal label strategy is used for every batch.
- dataset parameters: $n = 1000$ and $cm = \begin{pmatrix} 0.50 & 0.05 \\ 0.30 & 0.15 \end{pmatrix}$.

- operator time costs: t_{new}, t_{view}, $t_{overview}$, t_{single}, and t_{batch} to be the values measured for the JS-block application for any given $layout$.

We assume the goal of selection by the rank method rm is FN. Thus, one can verify that given the data confusion matrix $cm = \begin{pmatrix} 0.50 & 0.05 \\ 0.30 & 0.15 \end{pmatrix}$, the rank method precision of the BiPart rank method is equal to $\frac{9a}{2a+7}$. For $a \in <0, 0.25, 0.5, 0.75, 1>$, the corresponding rank method precisions are thus (approximately) $<0, 0.3, 0.56, 0.8, 1>$.

Ten repeated simulation trials with different random seeds are run for each combination of simulation parameters.

Result and Analysis

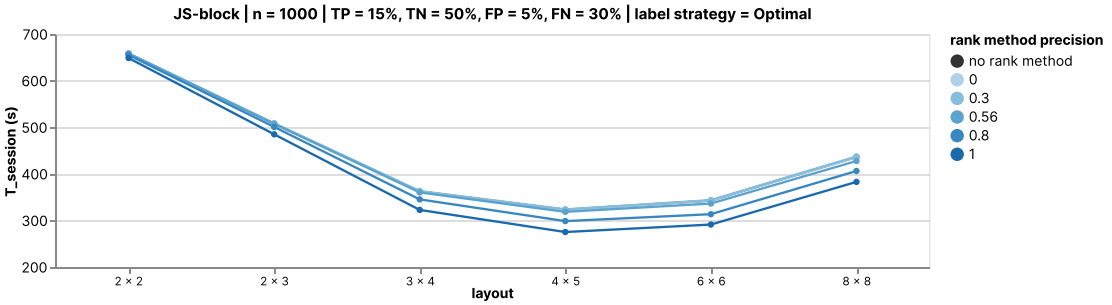


Figure 5.10: Variations of rank method precision: 5 conditions of rank method precision are simulated, including 0, 0.3, 0.56, 0.8, and 1. An additional condition called "no rank method" is simulated where the dataset is randomly ordered. The curves of "no rank method" and rank methods with precision 0 and 0.3 heavily overlap. The starting point of the vertical axis is 200 instead of 0 to highlight the differences in vertical positions.

Fig. 5.10 shows the simulation result. Compared with not using rank methods, introducing the rank method reduces the time cost. It can be seen that for all the layouts, when the rank method precision increases, the session time cost is consistently reduced. Asides, the benefit of having rank methods with high accuracy is more prominent for layouts with larger n_{batch} because the opportunities of using batch edit commands created by good rank methods are more useful for large n_{batch}. The result provides the design implication that for QA interfaces, **it is beneficial to integrate rank methods with high precision to promote the use of batch edit commands**.

5.8 Summary of the Simulation Model

The simulation model described above is intended to be used by UI specialists. UI specialists can use the model for different applications by specifying the simulation parameters. Table 5.1 summarizes all the simulation parameters discussed in the previous sections.

Table 5.1: The parameters of our simulation model. Common mathematical notations are used to denote the data types, such as $\mathbb{N}$ for integers, $\mathbb{R}$ for real numbers, + for > 0, 0+ for ≥ 0, $[a, b]$ for a real range.

Parameter	Data Type	Estimation	Description
n	$\mathbb{N}^+$	external	The number of data objects.
n_{batch}	$\mathbb{N}^+$	observed	The number of data objects of each batch.
cm	$[0,1]_{c\times c}$	external	Confusion matrix of default labels for the data to be quality-assured.
t_{new}	$\mathbb{R}^+$	captured	The average time cost to activate the "new batch" command to fetch the next batch of data objects.
t_{view}	$\mathbb{R}^+$	captured	The average time cost to view a data object and its label in a grid cell in the editing panel (including the viewing cost of the context panel).
$t_{overview}$	$\mathbb{R}^+$	captured	The average time cost to overview all data objects in the editing panel (including the viewing cost of the context panel).
t_{single}	$\mathbb{R}^+$	captured	The average time cost to activate a command to change the label of a data object in the editing panel.
t_{batch}	$\mathbb{R}^+$	captured	The average time cost to activate a batch edit command to change the labels of all the data objects in the editing panel.
u_{bcsl}	$[-1,1]$	observed	User skill level in using batch edit commands.
u_{bcsu}	$[0,1]$	observed	User strategy uncertainty in using batch edit commands.
$cmds$	list	observed	The list of UI commands to be simulated.
cmd	string	observed	A predefined UI command in the $cmds$, e.g.,"SingleEdit" and "AllPositive".
rm	$[0,1]_{c\times c}$	observed	A rank method defined by a selection matrix.

Some of these parameters are "external", typically independent of the QA process. For example, before the QA process, there should be a process of applying an ML model to generate a collection of labeled data objects to be checked by the QA process. Hence, the total number of data objects (n) and the normalized confusion matrix (cm) of the ML predictions should be known or estimable to UI specialists who conduct the simulation.

UI specialists who are to optimize a QA UI typically have some knowledge about the users who use the UI to perform the QA tasks and an essential experience for hypothesizing what design may be appropriate or inappropriate. We use "observed" to categorize parameters to be defined by UI specialists based on their observation of the data objects, users, tasks, and available rank methods and their likely effectiveness in the application context. UI specialists do not need to fix any parameters to a single value or setting. The benefit of using simulation is that UI specialists can consider many options.

For example, given some example data objects to be quality-assured, UI specialists can normally judge whether the display size of the data objects in the editing panel is too small or too large. Meanwhile, UI specialists may be uncertain if concurrently displaying 4×5 data objects is better than a 5×7 layout. UI specialists can simulate both scenarios by assigning different values to the parameter for batch size (n_{batch}). Similarly, UI specialists may consider having users with different QA experiences and at different skill levels. UI specialists can simulate different options by assigning different values to those user-related parameters (u_{bcsl} and u_{bcsu}). Choosing parameters related to the rank methods requires more technical knowledge. UI specialists may select a few relatively effective rank methods and determine their selection matrices (rm).

In Table 5.1, there are a set of operator time costs. Some of these operator time costs are relatively generic, such as t_{new}, which is the average time for pressing a button to fetch a new batch of data objects. They are relatively easy to measure and can be reused across different QA processes. Some of them are highly application-dependent, such as t_{view}, which is the average time for viewing a data object and its default label predicted by an ML model.

In general, we recommend that UI specialists conduct experiments to capture a few trials of using a QA UI. The timing parameters can be estimated based on captured data one by one, starting from simple ones, such as t_{new}, which are largely application-independent. While we focus on the quality assurance of classification labels, this approach can be adapted to other quality assurance tasks for model predictions (e.g., point cloud segmentation, time series forecasting) as long as user routine can be modeled.

5.9 Discussion

Our simulation approach attempts to provide an alternative to efficiently evaluate and compare design variations of quality assurance interfaces in data extraction and data labeling. As our approach can be seen as an application of the Keystroke-Level Model for quality assurance tasks in grid-based interfaces, we bear similar limitations as KLM. The following reflects on the limitations of our approach and future work directions.

Relaxing assumptions: Our modeling utilize several assumptions, including that the user makes no mistakes, the machine is fast enough, and the user operations are sequential. Future work may extend the simulation models to relax these assumptions. By assuming the user makes no mistakes in carrying out the operations, we are essentially estimating a lower bound of the task completion time. In the future, one may model the time cost caused by users' misuses of commands in the interface and the effort to correct such errors. Latency of the machine operations and parallelized human and machine operations may be considered when estimating the task completion time. Other factors, such as modeling user fatigue, which leads to time-varying performance, may also improve the modeling accuracy.

Evaluation metric other than task completion time: This chapter focuses on simulating the evaluation metric of task completion time in QA interfaces. The reason is that QA processes are typically laborious for the user, making the task completion time a critical metric in this scenario. The goal of quality assurance is well-defined, i.e., fixing incorrect labels, which also makes the estimation by task completion time feasible. There are other criteria generally crucial in the evaluation of user interfaces, such as user satisfaction and difficulties in learning the UI. Prior work has studied using model-based evaluation for criteria beyond task completion time, such as using GOMS to evaluate learning time [133]. Future work may examine how to adapt these criteria in the context of simulation-based evaluation for QA.

Interface designs other than grid-based: Our modeling mainly concerns the grid-based interfaces in the quality assurance of machine-predicted classification labels. These interfaces are based on buttons and menus, and the number of states of the interface is not too many. For QA interfaces with interaction mechanisms beyond button-based interactions, such as a paint-based interface to quality-assure image mask predictions, the modeling and simulation of user operations will need more considerations.

Validate with more experiment trials from different users: In the modeling of operator time costs in Section 5.4.3, we use the experiment log from self-experimentation to demonstrate how to do the modeling. The implications of simulation results, such as the optimal layout, can be validated through future experiment logs. By gathering data from multiple users, we may investigate additional user-related aspects, such as the variation of the operator time costs among users. Note that future experiments are

for accumulating records to validate our simulation model's prediction accuracy on the task completion time. In production usage scenarios, it is unnecessary and undesirable for evaluators using our model to use user studies to validate all the simulation results, which would diminish the purpose of using model-based evaluation as an efficient evaluation approach.

5.10 Conclusion

In quality-sensitive applications of QA, the user needs to use QA interfaces intensively. In such scenarios, interface evaluation and optimization bring significant benefits in terms of saved time for the user.

This chapter presents using model-based evaluation to assess and optimize QA interface design parameters through simulations. We focus on evaluating the time costs in QA as QA is typically labor-intensive. Our approach encompasses modeling the user's routine operations in QA process as a workflow, the user's operation time costs, the algorithmic assistance provided by the interface, and the user's strategy of using the interface with multiple commands available. We use data-driven modeling to estimate various operation time costs that are primarily data- and interface-dependent. We demonstrate that through the modeling and the corresponding simulation of the time cost with the proposed model, evaluation and optimization of QA interfaces are made possible.

Using QA usage scenarios, including data extraction from historical visualizations, we demonstrate the need for such an approach and the practical feasibility of using simulation to optimize the design of the interfaces. Specifically, we demonstrate the influence of various factors on the QA session time cost, including interface layout, application scenario, availability of interface functions, user's label strategy, default label accuracy, and rank method's precision. The simulations derived various findings, such as the dependency of the optimal layout condition with the application. As demonstrated, the simulation approach is generally applicable to quality assurance interfaces in Chapter 3's data extraction systems and Chapter 4's data labeling systems.

Model-based evaluation cannot replace user-centered approaches in designing and evaluating interfaces. It is impractical to expect the modeling approach to capture all the aspects of human factors. For ill-defined user tasks, such as open-ended data exploration, the evaluator may not be able to model the user's routine operations, making simulation infeasible.

When the user routine can be modeled, model-based evaluation is a cost-effective alternative for UI specialists to evaluate and optimize the interface. Simulation makes it possible to swiftly explore potential design parameters of interest and identify the influence of various factors on the QA time costs. It enables fast prototyping and iteration in early-stage interface development before expensive and time-consuming

user studies are conducted. With the simulations that can address many small but potentially costly design issues, we can focus our engagement with users on big design questions about data, ML models, workflows, and application contexts.

"Study without reflection is a waste of time. Reflection without study is dangerous." (translated from Chinese)

— Confuscius

6 Conclusion

Historical visualizations encode precious historical datasets. We start with the scenario of data extraction from historical visualization images in Chapter 3. During the process, we observe the connection between data extraction and data labeling. We investigate the general scenario of data labeling in supervised machine learning in Chapter 4. In Chapter 5, we conduct a model-based evaluation for quality assurance interfaces in data extraction and data labeling. The following summarizes the contributions and future work directions.

6.1 Contributions

We summarize the contributions of this thesis by revisiting the three research objectives in Chapter 1.

6.1.1 Efficiency and Effectiveness

The motivating scenario of the thesis is data extraction from historical visualizations. Manual data extraction is laborious. Meanwhile, data extraction can hardly be fully automated due to historical visualizations' diverse designs, image defects, and the data extraction's low error tolerance. To support efficient and effective data extraction (Section 1.2 **O1**), Chapter 3 proposes an interactive machine learning approach to join human and machine strengths.

We propose a generic interactive machine learning approach to data extraction from historical visualizations. It involves converting data extraction to classification tasks and providing machine assistance to reduce user effort in classification tasks.

Section 3.3 demonstrates examples of converting data extraction tasks into classification tasks. The conversion relates data extraction to data labeling and paves the way to integrate machine assistance.

For the converted classification tasks, we introduce a generic workflow to save user effort through machine assistance. The machine raises single-choice questions for the user to answer. It learns from user answers to provide tentative answers and prioritize the questions most challenging for the machine. This forms an active learning process that features active sampling and default labeling modules, with incremental model updates.

Section 3.4 demonstrates an interactive system for data extraction from spatial data visualizations. It implements three data extraction workflows that are instances of the generic workflow.

It is hard to build a single pipeline that works for all the diverse historical visualizations. A more practical approach is to build a set of elementary workflows and enable the user to compose them into different pipelines depending on the data extraction goal. Section 3.6 designs a set of 18 reusable workflows for data extraction. The workflows are categorized into visual object detection, visual object organization, and visual variable measurement. We design data extraction pipelines using the workflows as building blocks and demonstrate the feasibility of our design by prototyping two of the pipelines.

We have evaluated the efficiency of our approach in extracting data through a simulation-based evaluation in Section 3.5. The simulation results show that our approach saves user operations. We have conducted a domain expert interview, which reveals the usefulness of data extraction in history research.

Our design of reusable workflows structures the design space of data extraction systems. Our generic approach provides a way to implement the workflows as machine-aided classification processes. Through our research, we contribute to the solving of various data extraction problems. We demonstrate that for data extraction problems, imperfect machine-learned models can still be used to assist end-users and improve their productivity.

6.1.2 Generalization

Chapter 3 converts data extraction to classification tasks, which are instances of data labeling tasks. It thus inspires us to investigate the generalization of machine assistance for data extraction into the broader scenarios (Section 1.2 **O2**). Chapter 4 investigates the generalization of Chapter 3's approach to data labeling.

Data labeling systems are heavily used in supervised machine learning projects. Meanwhile, different projects frequently require building custom data labeling systems to accommodate the project requirements. We analyze reusable modules in data

labeling workflows. Through the coding of flowchart figures in the literature, we identify eight types of common modules: interactive labeling, data object selection, model training, feature extraction, default labeling, quality assurance, stoppage analysis, and label ideation. Each type of the modules can be implemented as a human computation procedure that requires user intervention or a machine computation procedure that is automatic.

Our framework of reusable modules structures the design space of data labeling systems. The identified modules generalize our design of machine assistance modules in Chapter 3.

Based on the conceptual framework of reusable modules in interactive machine learning systems for data labeling, we propose an authoring toolkit for developers to build machine-aided data labeling systems. The authoring toolkit enables the developer to visually configure data labeling workflows and develop data labeling systems with no code or less code. Section 4.5 demonstrates using the toolkit to build ten systems, including a data extraction system, in different scenarios of gathering benchmark datasets. A preliminary user study in Section 4.6 demonstrates that the toolkit enables developers to develop data labeling systems efficiently.

6.1.3 Evaluation and Optimization

While the machine assistance in Chapter 3 aims to reduce user effort, user intervention is required. In quality-sensitive applications, ML predictions need to be quality-assured by the user before the extracted dataset can be used confidently, which is a major source of time cost and user effort. Evaluation and optimization of interactive intelligent systems are thus necessary to reduce user effort. Meanwhile, conventional user studies are time-consuming and costly. Chapter 5 investigates an alternative swift evaluation approach.

In Chapter 5, we propose using model-based evaluation and optimization for quality assurance interfaces of data extraction and data labeling systems (Section 1.2 **O3**). We model the user and machine operation sequences in accomplishing quality assurance tasks. We consider factors in the quality assurance processes, especially user-related factors, including user operation time costs and the user's label strategy. The modeling facilitates simulation and estimation of task completion time in quality assurance processes.

We demonstrate using simulation to evaluate quality assurance interfaces in data extraction and aerial image classification. Through simulation, we investigate the influence of various factors on the task completion time, including the quality assurance interface layout, application scenario, availability of interface functions, user's label strategy, default label accuracy, and the ranking of data objects. We demonstrate optimizing the interface layout using simulation. Compared with conventional user

studies, the simulation approach provides a lightweight method for evaluators to compare design variations of label quality assurance interfaces.

6.2 Future Work

This section discusses the limitations of the work in the thesis and future research directions. We outline potential long-term projects and immediate short-term improvements corresponding to the three technical chapters:

1. **Develop an ultimate data extraction toolkit**: Section 3.6 envisions that by implementing a collection of some twenty data extraction workflows, we can compose them to handle various data extracting scenarios. We sketch a to-do list for this aim.

    (a) **Improve implementation coverage**: Chapter 3's implementation covers less than half of the data extraction workflows we design in Section 3.6. To engineer an ultimate data extraction toolkit, it is necessary to increase the implementation coverage. Additionally, as Section 3.6 has discussed, to handle diverse historical visualizations, it may be necessary to implement multiple instances for each workflow. For example, tracking-based object detection workflow (a_4 in Section 3.6.1) is useful for extracting lines in line charts. While the toolkit can provide an implementation of this workflow as an interactive machine learning process, to handle edge cases that this implementation fails, it is also necessary to implement a fallback option where the user can manually track the lines.

    (b) **Refine the design space**: The set of workflows we design in Section 3.6 is a design space for data extraction systems. It is critical to reflect on whether our design space should be revised throughout implementing more data extraction workflows. While we have conceptually checked the design space's expressiveness over a collection of historical visualizations, there may be edge cases of historical visualizations that we have missed. When approaching the goal of developing an ultimate data extraction toolkit, future research needs to decide what additional workflows is needed when encountering historical visualizations that cannot be handled. In addition to adding more workflows when encountering edge cases, it may also be helpful to consult research on visualization design space and visualization grammar [276]. Data extraction is the reverse process of data visualization. By consulting the design space of visualization, one may analytically reflect on whether our design space of data extraction systems needs revision.

(c) **Support multiple levels of reuse**: The toolkit should support the reuse of data extraction functionalities at multiple levels for different user groups. It should provide a collection of readily available data extraction software for common types of visualizations (e.g., bar chart data extraction pipeline) to support standard data extraction needs out of the box. For users with customization needs, the toolkit should provide reusable implementations at different abstraction levels to support coarse or fine-grained control of the customization. The toolkit should provide a collection of implementations of data extraction workflows (e.g., rectangular visual object detection workflow) that can be assembled into customized data extraction pipeline. It should also provide a collection of common modules in data extraction workflows (e.g., active sampling and default labeling) that can be assembled into data extraction workflows.

(d) **Support varied accuracy and automation requirements**: We have exclusively focused on the scenarios where the accuracy of data extraction is critical. There can be other scenarios that do not require high accuracy but require a high level of automation. Generally, it calls for an effort to investigate the best practices for combining user and machine involvements in data extraction scenarios that span the continuous requirement spectrum between high accuracy and high automation.

(e) **Investigate user interventions other than button clicking**: Chapter 3 mainly considers user intervention through button clicking. The user clicks buttons to edit the labels of data objects and clicks buttons to confirm labels. One may come up with a design variation of our interactive machine learning approach where the user uses freeform drawing (instead of button clicking) to label data objects in the visualization image, and the machine learns a segmentation model (instead of a classification model) to assist the user in the drawing. Our rationale for focusing on button clicking is that it is easier and more efficient than other user operations, such as drawing freeform shapes. It may be worthwhile to investigate alternative forms of user interventions, such as freeform drawing for data extraction, and validate our rationale by comparing the usability of these design variations.

While supporting these functionalities, we expect the major challenge is implementing the preprocessing and postprocessing modules for each workflow. The implementation of these two types of modules are heavily dependent on the workflow, while the others are loosely coupled with the workflow. Based on our own experience, developing these two steps, typically as image processing algorithms, needs trial and error.

Another challenge that requires attention and engineering skills is the speed

optimization of algorithmic modules. Interactive systems should support real-time interaction to avoid frustrating users. Meanwhile, a data extraction system usually involves various algorithms for machine learning and image processing, which can be computationally expensive. The developer needs to design the algorithmic modules to execute fast enough or run in the background without blocking the user. In the worst cases where heavy computation is unavoidable, the developer may batch the heavy computations to the beginning of the workflow so that the user only needs to wait once at the beginning.

2. **Develop application-specific software with data extraction function**: Through the interview with domain experts in Section 3.7, we observe that while data extraction is deemed useful, it typically needs further processing to derive output directly usable by the historians. For example, for studying the spatial correlation of landmarks, the location of the visual objects needs to be extracted. Then, the location data need further processing to generate statistics and possibly redesigned visualizations for historians to analyze the spatial correlation. While we may provide a data extraction software for them to extract data and then send the data to another piece of software for further analysis, the users may need to move back and forth between multiple pieces of software in this case, which is laborious and undesirable. Thus, it may be beneficial to build application-specific software that embeds the data extraction functions and directly address domain experts' needs. To fulfill this requirement, we need to redesign a software architecture of data extraction functions so that they can be easily embedded in application-specific software and interoperate with other preprocessing or postprocessing functionalities.

3. **Develop a general-purpose toolkit for interactive machine learning**: Developing interactive machine learning systems is generally challenging [123], as they often target intricate usage scenarios and thus require complex functionalities. It calls for infrastructures to accelerate the development, like Tensorflow [1] and PyTorch [206] to the ML community. Chapter 4 demonstrates a toolkit for authoring data labeling systems. It generalizes the data extraction approach in Chapter 3. A future direction is to generalize the toolkit further to be a toolkit for authoring interactive machine learning systems. Our toolkit currently does not capture topics in IML irrelevant to data labeling, such as model explanation. Thus, the building blocks in the toolkit need to be extended to step toward generalization. Achieving this goal requires identifying and implementing additional common modules in IML systems.

4. **Refine the simulation model**: Chapter 5 uses model-based evaluation for data extraction and data labeling systems. Our simulation model is based on simplifi-

cation assumptions, such as assuming the user does not make mistakes when carrying out operations. A direction of future work is to relax the assumptions listed in Section 5.3.4. For the assumptions related to the user, experiments with real users are necessary to either validate the assumption or derive alternative models that better capture the user behavior.

5. **Validate simulation results with user experiment data**: The simulation model in Chapter 5 is essentially a predictive model of task completion times. The implications of simulation results in Chapter 5 need to be validated with observations in future user experiments to examine whether the predictions are accurate or the model needs to be fine-tuned. Meanwhile, we note that our simulation model is for the evaluator to rule out sub-optimal designs quickly. Thus, we expect that after future work on refining the simulation model and validating its accuracy, the evaluator should not frequently run user studies in production usage scenarios to validate model predictions. Otherwise, the cost of running the user studies for validation would diminish the purpose of using simulations for swift evaluation.

6. **Integrate model-based evaluation in the authoring toolkit**: Chapter 5's simulation approach aims to make it easier to evaluate quality assurance interfaces. We believe it beneficial to integrate the simulation functions into Chapter 4's authoring toolkit. This way, the UI developer may develop a few design variations and try out the prototyped interfaces for a few trials to provide experiment data. A model-based evaluation routine may then automatically generate a report to inform the developer which design variation is the best according to the simulated task completion time. Through this integration, the iterative loop of development and evaluation can be shortened.

Appendices

A

Graph-Theoretic Constraints on Data Labeling Workflows

A.1 Notations

- $G = (V, E)$: a directed graph modeling a data labeling system.
- V: a set of nodes, each corresponding to a software module, or initialization, decision, and exit node.
- E: a set of edges specifying the execution order.
- $type(v)$: the type of a node v, which takes one of the following values: initialization, process, decision, exit.
- $fun(v)$: the function of a node v. When $type(v)$ takes value in { initialization, decision, exit }, $fun(v) = type(v)$. When $type(v) = process$, $fun(v)$ takes one of the following values: interactiveLabeling, dataObjectSelection, modelTraining, featureExtraction, defaultLabeling, qualityAssurance, stoppageAnalysis, labelIdeation.
- $input(v)$: the set of input states to a node v.
- $output(v)$: the output state of a node v.
- $Exec(G)$: the set of all the directed walks on the graph from the initialization node to the exit node.

A.2 Constraints

Valid Flowchart:

- The graph contains no parallel edges (i.e., E is not a multi-set).
- The graph contains one initialization node (denoted as v_s).

$$\exists! v(v \in V \wedge type(v) = initialization)$$

- The graph contains one exit node (denoted as v_e).

$$\exists! v(v \in V \wedge type(v) = exit)$$

- A process node has outdegree 1.

$$\forall v(v \in V \wedge type(v) = process \rightarrow deg^+(v) = 1)$$

- A decision node has outdegree 2.

$$\forall v(v \in V \wedge type(v) = decision \rightarrow deg^+(v) = 2)$$

- An initialization node has indegree 0 and outdegree 1.

$$\forall v(v \in V \wedge type(v) = initialization \rightarrow deg^+(v) = 1 \wedge deg^-(v) = 0)$$

- An exit node has outdegree 0.

$$\forall v(v \in V \wedge type(v) = exit \rightarrow deg^+(v) = 0)$$

- All the nodes can be reached from the initialization node. The exit node can be reached from all the nodes.

$$\forall v(v \in V \rightarrow \exists exec(exec \in Exec(G) \wedge exec = (v_s, ..., v, ..., v_e)))$$

- No self-loops.

$$\forall v(v \in V \rightarrow (v, v) \notin E)$$

Input Initialized:

$$\forall exec = (v_s, ..., v_e) \in Exec(G) \forall v_i \in exec \forall input \in input(v_i)$$

$$\exists j(j < i \wedge output(v_j) = input)$$

No Redundancy:

- After a module is visited, it should not be revisited until at least one of its inputs' value has been changed.

$$\forall exec = (v_1, ..., v_k) \in Exec(G)$$

$$((\exists 1 \leq i < j \leq k(v_i = v_j)) \rightarrow$$

$$(\exists i < l < j(output(v_l) \in input(v_j))))$$

- After a module is visited, its output(s) should be used by a module.

$$\forall exec = (v_1, ..., v_k) \in Exec(G)$$

$$\forall 1 \leq i \leq k(output \in output(v_i) \rightarrow$$

$$(\exists i < j \leq k(output \in input(v_j) \wedge \nexists i < l < j(output \in output(v_l)))))$$

Involve Labeling:

$$\forall exec = (v_s, ..., v_e) \in Exec(G) \exists i(fun(v_i) = interactiveLabeling)$$

B

User Study Instructions for the Toolkit

The following are the instructions we gave the participants for the 4 labeling system development tasks in Section 4.6.

B.1 Task 1 (Image Segmentation)

Please create a labeling system for image segmentation according to the following specification of the labeling system's functionality. Please do not use OneLabeler's built-in templates for this task.
Specification:

1. In the image segmentation system, data objects are iteratively selected for annotators to label. 1 data object is randomly selected each time.
2. The interface displays the 1 data object in a single object display.
3. After annotating the 1 data object, the system checks whether all the data objects are labeled.
4. If all the data objects are labeled, the labeling STOPS.
5. Otherwise, goto step 1.

Please tell the experimenter when you have finished.

B.2 Task 2 (Image Classification)

Based on the previous image segmentation workflow in Task 1, please modify the workflow to create another labeling system for image classification according to the

following specification of the labeling system's functionality.
Specification:

1. In the image classification system, data objects are iteratively selected for annotators to label. 16 data objects are randomly selected each time.
2. The interface displays the 16 data objects in a grid with 4 rows and 4 columns.
3. After annotating the 16 data objects, the system checks whether all the data objects are labeled.
4. If all the data objects are labeled, the labeling STOPS.
5. Otherwise, goto step 1.

Please tell the experimenter when you have finished.

B.3 Task 3 (Machine-Aided Image Classification)

Based on the previous image classification workflow in Task 2, please modify the workflow to create another labeling system for image classification according to the following specification of the labeling system's functionality.
Specification:

1. At the beginning, SVD features are extracted for the data objects.
2. Then, data objects are iteratively selected for annotators to label. 16 data objects are randomly selected each time.
3. The selected data objects are then assigned default labels by a decision tree classifier.
4. The interface displays the 16 data objects in a grid with 4 rows and 4 columns.
5. After annotating the 16 data objects, the system checks whether all the data objects are labeled.
6. If all the data objects are labeled, the labeling STOPS.
7. Otherwise, the decision tree classifier used for default labeling is updated by retraining.
8. Then, goto step 2.

Please tell the experimenter when you have finished.

B.4 Task 4 - Option 1 (Customize Data Type)

Based on the given image classification workflow template, please modify the workflow to create another labeling system for webpage classification according to the following specification of the labeling system's functionality.
Specification:

1. In the webpage classification system, data objects are iteratively selected for annotators to label. 2 data objects are randomly selected each time.

2. The interface displays the 2 data objects in a grid with 1 row and 2 columns.

3. After annotating the 2 data objects, the system checks whether all the data objects are labeled.

4. If all the data objects are labeled, the labeling STOPS.

5. Otherwise, goto step 1.

The webpage data type is not currently supported in OneLabeler. Before you start to build the labeling system in the visual programming interface, you will need to first create the new webpage data type with some coding. To create this new data type, please follow the instructions below.

1. Enter the "./client" folder of OneLabeler's source code.

2. Execute "npm run customize" in the command line, and choose suitable options in the command-line interface to create template code files for the webpage data type.

3. Revise template code files to create the webpage data type.

The webpage data type should satisfy this specification:
When using a grid to display webpage data objects, each grid should display the webpage content. Each webpage data object should fill the grid that contains it.
Additional Tips:

- You may first read the "README.md" file in the created template code files, which gives you an overview of what the template code files are.

- When editing the code, you may refer to the HTML iframe tag API[1] if you are unfamiliar with it.

[1] https://developer.mozilla.org/docs/Web/HTML/Element/iframe

- In this task, you will not need to edit any code files other than the files created by the “npm run customize” command.
- You may refer to the “Customization” section of the documentation website.
- If you do not have web development experience, you may ask the experimenter to help you with the programming part.

Please tell the experimenter when you have finished.

B.5 Task 4 - Option 2 (Reproduce Your Tool)

Please use OneLabeler to reproduce your labeling system. The reproduction should have major functionalities similar to your labeling system but does not need to be the same. For this task, you may ask the experimenter for help.

C

Estimating Operator Time Costs in Quality Assurance Interfaces

C.1 Experiment

We aim to estimate t_{new}, t_{view}, and t_{single} as outlined in Section 5.4.3. To facilitate the estimation, we need to gather observations of $(T_{round}, N_{new}, N_{view}, N_{single})$ through the experiment. We vary N_{new}, N_{view}, N_{single} in experiment trials and measure time cost T_{round}. N_{new} is the number of times the user activate the "new batch" command to request a new batch of data objects to quality-assure. N_{view} is the number of times the user carries out the "viewing" action to comprehend the visual representation of a single data object. N_{single} is the number of times the user activates the "single edit" command to change the label of a data object.

C.1.1 Apparatus

As the developer of the interfaces, the user is familiar with all the functionalities of the interface. The user has much experience using the interfaces before the experiment and represents the user group with high expertise in using the interfaces. The experiment was conducted on a laptop with a 15-inch 3:2 screen. The screen resolution is 2560 × 1600. The screen is viewed by the participant from approximately 80 cm away. The participant used a physical mouse to conduct the tasks. The DPI of the mouse is 2000 dots per inch.

The experiment is carried out for three quality assurance interfaces as shown in Fig. 5.2(a - c). The QA interfaces are all implemented as web applications that run in a browser. We implemented a browser plugin to log observations of T_{round}. One interface is for data extraction from visualization images that requires quality assurance

Table C.1: Experiment design: The number of repeated trials conducted for each combination of the independent variables: application, grid layout, number of data objects.

Application	Layout	#Objects	#Trials
JS-block	1 × 1	0	1
JS-block	1 × 1	1	20
JS-block	1 × 2	0	1
JS-block	1 × 2	1	15
JS-block	1 × 2	2	15
JS-block	2 × 2	0	1
JS-block	2 × 2	2	6
JS-block	2 × 2	4	15
JS-block	2 × 3	0	1
JS-block	2 × 3	3	10
JS-block	2 × 3	6	10
JS-block	3 × 4	0	1
JS-block	3 × 4	4	10
JS-block	3 × 4	8	10
JS-block	3 × 4	12	10
JS-block	4 × 5	0	1
JS-block	4 × 5	10	4
JS-block	4 × 5	15	6
JS-block	4 × 5	20	6
JS-block	6 × 6	0	1
JS-block	6 × 6	12	4
JS-block	6 × 6	18	3
JS-block	6 × 6	24	5
JS-block	6 × 6	30	6
JS-block	6 × 6	36	7
JS-block	8 × 8	0	1
JS-block	8 × 8	32	4
JS-block	8 × 8	48	5
JS-block	8 × 8	64	3

for JS-block and JS-grouping (see Fig. 5.1(b)). The other interface is for solar panel detection from remote sensing images that requires quality assurance for SP-image (see Fig. 5.1(a)). Each quality assurance interface presents a grid panel with each grid cell corresponding to a data object.

C.1.2 Procedure

The experiment is carried out through self-experimentation by the author for proof-of-concept. We carried out in total 332 trials, as shown in Table C.1 and Table C.2 for the three different applications, with different grid layouts and different numbers of grid cells shown in the interfaces.

Before the beginning of each trial, the browser plugin adds a white overlay to the interface and highlights the location of the grid cells. After the user clicks the overlay, the timer of the trial starts. The user quality-assures the labels with the provided

Table C.2: Experiment design continued: The number of repeated trials conducted for each combination of the independent variables: application, grid layout, number of data objects.

Application	Layout	#Objects	#Trials
JS-grouping	1 × 1	0	1
JS-grouping	1 × 1	1	20
JS-grouping	2 × 3	0	1
JS-grouping	2 × 3	3	7
JS-grouping	2 × 3	6	6
JS-grouping	3 × 4	0	1
JS-grouping	3 × 4	4	3
JS-grouping	3 × 4	8	3
JS-grouping	3 × 4	12	3
JS-grouping	4 × 5	0	1
JS-grouping	4 × 5	10	5
JS-grouping	4 × 5	15	5
JS-grouping	4 × 5	20	5
JS-grouping	6 × 6	0	1
JS-grouping	6 × 6	12	5
JS-grouping	6 × 6	24	5
JS-grouping	6 × 6	36	5
JS-grouping	8 × 8	0	1
JS-grouping	8 × 8	24	1
JS-grouping	8 × 8	32	4
JS-grouping	8 × 8	48	5
JS-grouping	8 × 8	64	3
SP-image	1 × 1	0	1
SP-image	1 × 1	1	5
SP-image	2 × 3	0	1
SP-image	2 × 3	3	4
SP-image	2 × 3	6	4
SP-image	3 × 4	0	1
SP-image	3 × 4	4	4
SP-image	3 × 4	8	4
SP-image	3 × 4	12	4
SP-image	4 × 5	0	1
SP-image	4 × 5	10	4
SP-image	4 × 5	15	4
SP-image	4 × 5	20	4
SP-image	5 × 8	0	1
SP-image	5 × 8	16	4
SP-image	5 × 8	24	4
SP-image	5 × 8	32	4
SP-image	5 × 8	40	4

functionality of the interface. Once the user clicks the confirm button, the timer stops, and the time cost of the trial is logged. To alleviate the learning effect, each trial is conducted twice, and the time is logged only when the trial is conducted the second time. To simulate different levels of default label accuracies, the default labels of data objects are programmatically randomized. For example, for the binary classification case, given an accuracy acc, the program randomly selects $round(acc * n_{batch})$ data objects to flip their true labels.

C.2 Initial Estimations of Operator Time Costs

Using the experiment data, we estimate t_{new}, t_{view}, t_{single} with multiple linear regression. Specifically, we go through the following estimation procedure for each combination of (application, layout). We fit an individual model for each combination of (application, layout) under the consideration that the application and layout may significantly influence the operation time costs.

Given (application, layout), assume T_{round} follows a multiple linear model with regards to N_{new}, N_{view}, N_{single} as $T_{round} = N_{new}t_{new} + N_{view}t_{view} + N_{single}t_{single}$ in Equation 5.1. We fit a multiple linear regression model for T_{round} to solve for t_{new}, t_{view} and t_{single}. The standard error, R^2, and significance of the fitting are computed. Table C.3 and Fig. C.1 show the estimation result.

C.3 Modeling and Reestimation of Operator Time Costs

In the following, we introduce a process of smoothing the operator time costs estimated in Section C.2. We fit curves for the operator time costs and use the values on the curves to substitute the initial estimations of the operator time costs in Section C.2.

C.3.1 Model t_{new}

Goal Reestimate t_{new} as a function of interface parameter n_{batch} to smooth the estimation.

Procedure There are 19 combinations of (application, layout) and thus 19 data points (n_{batch}, t_{new}) to use. We fit a model used for all three applications. We used the Online Nonlinear Regression Tool [282] to produce a set of candidate model functions $t_{new} = f(n_{batch})$.

- With the number of parameters = 2, we get the top 3 functions being
  - $y = \frac{ax}{x+b}$ with $R^2 = 0.9205$
  - $y = ae^{b/x}$ with $R^2 = 0.8989$

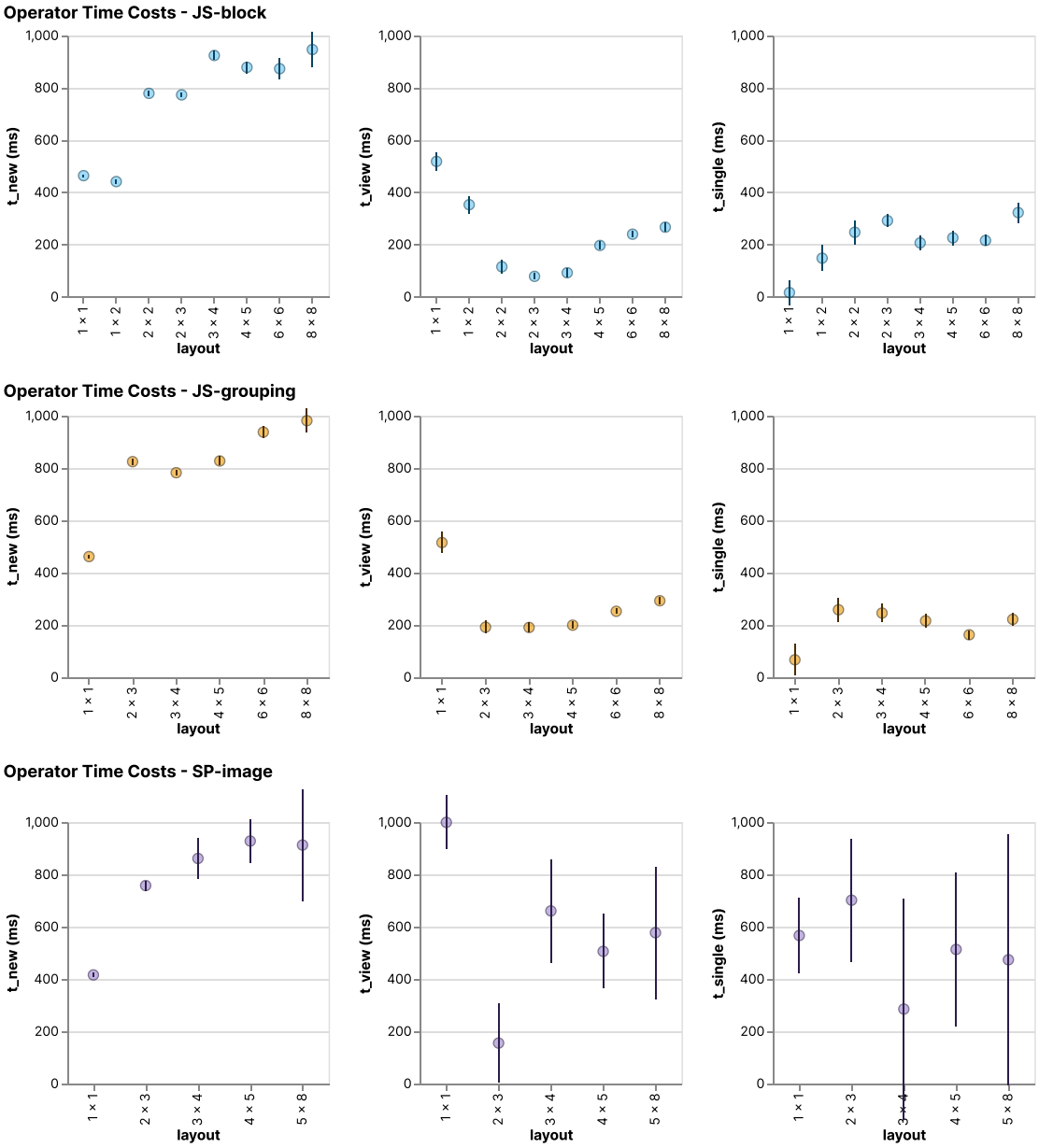


Figure C.1: Initial estimations of operator time costs: The estimated operator time costs t_{new}, t_{view}, and t_{single} in relation to the grid layouts for all the applications. The error bars show the estimated standard error of the estimation. It can be seen that as the number of grid cells grows, t_{new} increases, t_{view} first increases then decreases, t_{single} first increases then stays stable.

Table C.3: Initial estimations of operator time costs: The estimated time costs of unit user operations using multiple linear regression. For each estimation, the estimated standard error is shown in the parenthesis. The statistical significance is marked with ** meaning $p \leq 0.01$ and * meaning $p \leq 0.05$. R^2 columns show the R-squared of the entire linear model $T_{round} = N_{new}t_{new} + N_{view}t_{view} + N_{single}t_{single}$. It can be seen that for all the experimented applications and grid layouts, N_{new} contributes linearly to T_{round}. In most cases, N_{view} and N_{single} contributes linearly to T_{round}.

Application	**Layout**	$\mathbf{t_{new}}$ (ms)	$\mathbf{t_{view}}$ (ms)	$\mathbf{t_{single}}$ (ms)	$\mathbf{R^2}$
JS-block	1 × 1	463 (± 5) **	518 (± 35) **	16 (± 47)	0.998
JS-block	1 × 2	441 (± 8) **	352 (± 33) **	148 (± 49) **	0.993
JS-block	2 × 2	778 (± 10) **	115 (± 25) **	247 (± 46) **	0.997
JS-block	2 × 3	773 (± 7) **	78 (± 12) **	291 (± 24) **	0.999
JS-block	3 × 4	923 (± 18) **	92 (± 16) **	206 (± 28) **	0.993
JS-block	4 × 5	878 (± 23) **	196 (± 15) **	226 (± 28) **	0.997
JS-block	6 × 6	873 (± 39) **	239 (± 12) **	215 (± 21) **	0.995
JS-block	8 × 8	946 (± 68) **	266 (± 18) **	322 (± 38) **	0.997
JS-grouping	1 × 1	461 (± 7) **	515 (± 40) **	66 (± 59)	0.997
JS-grouping	2 × 3	824 (± 12) **	191 (± 25) **	258 (± 45) **	0.998
JS-grouping	3 × 4	782 (± 10) **	190 (± 19) **	245 (± 35) **	0.999
JS-grouping	4 × 5	827 (± 17) **	199 (± 14) **	215 (± 25) **	0.998
JS-grouping	6 × 6	937 (± 22) **	252 (± 10) **	162 (± 18) **	0.998
JS-grouping	8 × 8	981 (± 46) **	292 (± 14) **	221 (± 25) **	0.998
SP-image	1 × 1	416 (± 9) **	999 (± 103) **	566 (± 145) *	0.999
SP-image	2 × 3	757 (± 20) **	155 (± 151)	701 (± 234) *	0.997
SP-image	3 × 4	861 (± 78) **	660 (± 198) **	285 (± 423)	0.976
SP-image	4 × 5	928 (± 84) **	506 (± 142) **	513 (± 295)	0.988
SP-image	5 × 8	912 (± 214) **	577 (± 252) *	473 (± 479)	0.976

  - $y = a + b/x$ with $R^2 = 0.8630$

- With the number of parameters = 3, we get the top 3 functions being
  - $y = a + b/x + c/x^2$ with $R^2 = 0.9381$
  - $y = ax + \frac{bx}{x+c}$ with $R^2 = 0.9231$
  - $y = \frac{1}{ax^b+c}$ with $R^2 = 0.9225$
- With the number of parameters = 4, we get the top 3 functions being
  - $y = a + b/x + c/x^2 + d/x^3$ with $R^2 = 0.9486$
  - $y = a + b/x + c/x^2 + dx$ with $R^2 = 0.9486$
  - $y = a + b/x + c/x^2$ with $R^2 = 0.9381$

It can be seen that $y = a + b/x + c/x^2$ is reoccurring. It is the best for parameters = 3 and the third-best for parameters = 4. Adding additional terms generate the best function for parameters = 4. Removing the c/x^2 term generates the third-best for parameters = 2.

We refer to the literature for more function options. The "new batch" button clicking is related to the object-pointing task concerned in Fitts' law [90]. Fitts' law suggests that the object-pointing time cost can be modeled as $T = a + blog(\frac{A}{W} + 1)$. W is the size of the target object, A is the distance from the initial cursor position to the target object, whereas a and b are parameters to be fitted which may depend on interface and apparatus settings.

To activate the "new batch" command, the user needs to move the cursor to the "new batch" button and click it. This procedure is similar to the Fitts' law setting, while a significant difference from typical Fitts' law experiment settings is that the distance to the target (the "new batch" button) is not well-defined. Because the last position of the cursor before the user decides to activate the "new batch" command can be an arbitrary point in the interface.

We adopt the following approximation to examine whether Fitts' law works for our scenario. Assume the aspect ratio of the grid cells does not change with the increase of n_{batch}. Denote the layout as $xf \times yf$ where x, y, f are integers. The maximal distance of points in the interface is thus $\sqrt{x^2 + y^2} f$. As f increases, the number of grid cells xyf^2 increases at the rate of f^2, and the maximal distance $\sqrt{x^2 + y^2} f$ increases at the rate of f. We approximate the distance A in Fitts' law by the maximal distance $\sqrt{x^2 + y^2} f$ and then represent it as $c\sqrt{n_{batch}}$ where c is an interface related constant. Thus, we get $t_{new} = a + blog_2(c\sqrt{n_{batch}} + 1)$. However, this function is not convex and hard to optimize. Therefore, we modify it to $t_{new} = a + blog_2(\sqrt{n_{batch}} + 1)$ by removing c.

In short, we additionally consider the model function $t_{new} = a + blog_2(\sqrt{n_{batch}} + 1)$ to see whether Fitts' law applies to our scenario.

Thus, we choose $y = a + b/x + c/x^2$ and $y = a + blog_2(\sqrt{x} + 1)$ to be the candidate function families for t_{new}. We fit the models with linear regression.

Outcome

- For $y = a + b/x + c/x^2$, the function fitted is:

$$t_{new} = 958.0085 - 1254.8737 \times 1/n_{batch} + 739.4750 \times 1/n_{batch}^2$$
$$\text{with adjusted } R^2 = 0.9303, SE = 49.4461$$

- For $y = a + blog(\sqrt{x} + 1)$, the function fitted is:

$$t_{new} = 280.0475 + 341.5430 \times log_2(\sqrt{n_{batch}} + 1)$$
$$\text{with adjusted } R^2 = 0.8004, SE = 83.6951$$

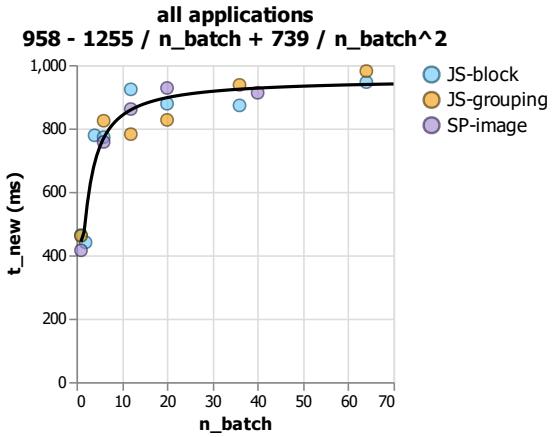


Figure C.2: The estimated model of t_{new} as a function of n_{batch} for the three applications using $y = a + b/x + c/x^2$.

It can be seen that the fitting of $y = a + blog(\sqrt{x}+1)$ suggested by Fitts' law performs worse than $y = a + b/x + c/x^2$ in terms of adjusted R^2 and SE. Thus, we decide to adopt:

$$\mathbf{t_{new} = 958.0085 - 1254.8737 \times \frac{1}{n_{batch}} + 739.4750 \times \frac{1}{n_{batch}^2}} \tag{C.1}$$

C.3.2 Reestimate t_{view} and t_{single} by t_{new} Model

Goal Reestimate t_{view} and t_{single} by removing the previous modeled t_{new} from the equations to possibly reduce the noise and make sure the original equations still approximately hold.

Procedure With the t_{new} modeled in the last sections, we reestimate t_{view} and t_{single} from the experiment data by removing the contribution of t_{new} as

$$\begin{bmatrix} t_{view} \\ t_{single} \end{bmatrix} = (X^T X)^{-1} X^T \begin{bmatrix} T_1 - N_{new,1}\, t_{new} \\ T_2 - N_{new,2}\, t_{new} \\ \cdots \\ T_n - N_{new,n}\, t_{new} \end{bmatrix} \tag{C.2}$$

where X is the N_{view}, N_{single} measured for the n trials

$$X = \begin{bmatrix} N_{view,1} & N_{single,1} \\ N_{view,2} & N_{single,2} \\ \cdots & \cdots \\ N_{view,n} & N_{single,n} \end{bmatrix}$$

and the t_{new} in the formula use modeled t_{new} values in the last section.

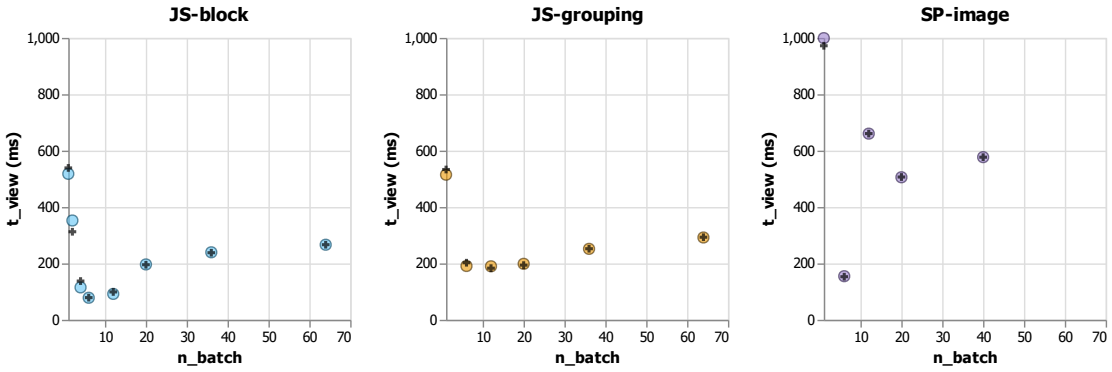


Figure C.3: t_{view} reestimation by t_{new}: The influence of reestimation to t_{view} as a function of n_{batch} for the three applications. The colored dots are original estimations by multiple linear regression. The black crosses are reestimations.

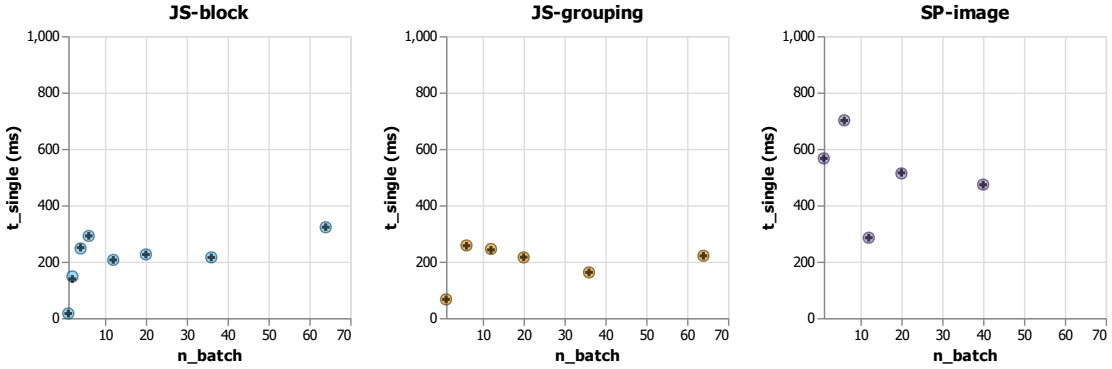


Figure C.4: t_{single} reestimation by t_{new}: The influence of reestimation to t_{single} as a function of n_{batch} for the three applications. The colored dots are original estimations by multiple linear regression. The black crosses are reestimations.

Outcome

- **For t_{view}**: The influence of the reestimation is shown in Fig. C.3. The first row is the initial estimations by multiple linear regression. The second row is after reestimation by putting t_{new} back.
- **For t_{single}** The influence of the reestimation is shown in Fig. C.4. The first row is the initial estimations by multiple linear regression. The second row is after reestimation by putting t_{new} back.

Comments For both t_{view} and t_{single}, reestimation makes little change to the data points.

- **For t_{view}**: t_{view} seems to follow a U-shaped curve for JS-block and edges. t_{view} for SP-image is very noisy and hard to interpret. The pattern of t_{view} is not simple. We decide the leave t_{view} at the moment.

- **For t_{single}**: One can also observe that the t_{single} for the two snow examples display almost a constant or weak linear pattern except for the first data point.

The actions of single edit may happen during and after that of view. The attribution of time to single edit and view may be less accurate when few data objects are sampled (i.e., when n_{batch} is small), and the number of single edit actions is sparse.

Compared to the data points' pattern in JS-block and JS-grouping, the second and third data points of t_{single} of SP-image appear to be anomalous. For the SP-image example, if we ignore the second and third outlying data points, we can observe a constant or weak linear pattern across the remaining data points.

It is reasonable to assume that the time taken for performing a single edit action is reasonably consistent within an application but inconsistent across different applications. Fig. C.4 evidences this while showing that it has a less complicated pattern than t_{view}.

The pattern of t_{single} is more straightforward than t_{view}. Thus, we decide to model it first.

C.3.3 Model Reestimated t_{single}

Goal Model the reestimated t_{single} to reduce noise.

Procedure We model t_{single} as a function of n_{batch}. There is no clear evidence that t_{single} is application-independent. Thus, we fit a model for each application. For JS-block, there are 8 data points. For JS-grouping, there are 6 data points. For SP-image, there are 5 data points. We used the Online Nonlinear Regression Tool [282] to produce a set of candidate model functions $t_{single} = f(n_{batch})$.

- With the number of parameters = 2, we get the top 3 functions being:
  - for JS-block:
    - $y = a + b/x$ with $R^2 = 0.7795$
    - $y = ae^{b/x}$ with $R^2 = 0.7049$
    - $y = \frac{ax}{x+b}$ with $R^2 = 0.6303$
  - for JS-grouping:
    - $y = a + b/x$ with $R^2 = 0.6940$
    - $y = ae^{b/x}$ with $R^2 = 0.6400$
    - $y = ax^{b/x}$ with $R^2 = 0.5938$
  - for SP-image:
    - $y = acos(bx)$ with $R^2 = 0.3560$

    - $y = a + ln(x)$ with $R^2 = 0.1385$
    - $y = ax^b$ with $R^2 = 0.1364$

- With the number of parameters = 3, we get the top 3 functions being:
  - for JS-block:
    - $y = \frac{1}{a+bx^c}$ with $R^2 = 0.8310$
    - $y = a + bx + c/x^2$ with $R^2 = 0.8240$
    - $y = a + b/x + c/x^2$ with $R^2 = 0.7939$
  - for JS-grouping:
    - $y = a + b/x + c/x^2$ with $R^2 = 0.9018$
    - $y = ae^{b/x+c}$ with $R^2 = 0.8927$
    - $y = ae^{b/x+cln(x)}$ with $R^2 = 0.8506$
  - for SP-image:
    - $y = ax^{bx^c}$ with $R^2 = 0.4980$
    - $y = acos(bx)$ with $R^2 = 0.3560$
    - $y = a + b/x + c/x^2$ with $R^2 = 0.3172$

- With the number of parameters = 4, we get the top 3 functions being:
  - for JS-block:
    - $y = a + bx + c/x + dln(x)$ with $R^2 = 0.9059$
    - $y = a + bx^{0.5} + cx + dx^{1.5}$ with $R^2 = 0.8574$
    - $y = a + bx + ((bx - c)^2 - d)^{0.5}$ with $R^2 = 0.8332$
  - for JS-grouping:
    - $y = acos(x + b) - ccos(2x + b) - dcos(3x + b)$ with $R^2 = 0.9957$
    - $y = a + b^{0.5} + cx + dx^{1.5}$ with $R^2 = 0.9804$
    - $y = a + bx + c/x + dln(x)$ with $R^2 = 0.9562$
  - for SP-image:
    - $y = a + b/x + c/x^2 + d/x^3$ with $R^2 = 0.7832$
    - $y = a + bsin(cx + d)$ with $R^2 = 0.6659$
    - $y = a + bx + c/x + d/x^2$ with $R^2 = 0.5589$

It can be seen that $y = a + b/x + c/x^2$ is reoccurring. It is the third-best for JS-block and SP-image and best for JS-grouping when parameters = 3. Removing the c/x^2 term generates the best for JS-block and JS-grouping when parameters = 2. Adding the d/x^3 term generates the best for SP-image when parameters = 4.

Thus, we choose $y = a + b/x + c/x^2$ to be a candidate function family for t_{single}.

Moreover, we noticed that for JS-block and JS-grouping, if we ignore the 1×1 layout, t_{single} seems to follow a constant or weak linear pattern. Thus, we also choose $y = a$ and $y = a + bx$ to be candidate function families for t_{single}

We refer to the literature for more function options. The single edit button clicking is somehow related to the decision-making task in Hick's law [120]. Hick's law suggests that the average reaction time to choose among n equally probable choices can be modeled as $T = blog_2(n+1)$ where b is a parameter to be fitted. If we assume the solved t_{single} is a time cost for the user to point to a grid cell and click it, t_{single} may be modeled as $t_{single} = a + blog_2(n_{batch}+1)$. The a term denotes the constant cost of clicking, while the $blog_2(n_{batch}+1)$ term denotes the decision time following Hick's law. Our assumption needs further investigation, as it is possible that the decision time should be included in t_{view} instead of t_{single}.

In summary, we fit 4 models, $y = a+b/x+c/x^2$, $y = a+blog_2(x+1)$, $y = a$, $y = a+bx$. We fit the models all with linear regression.

Outcome

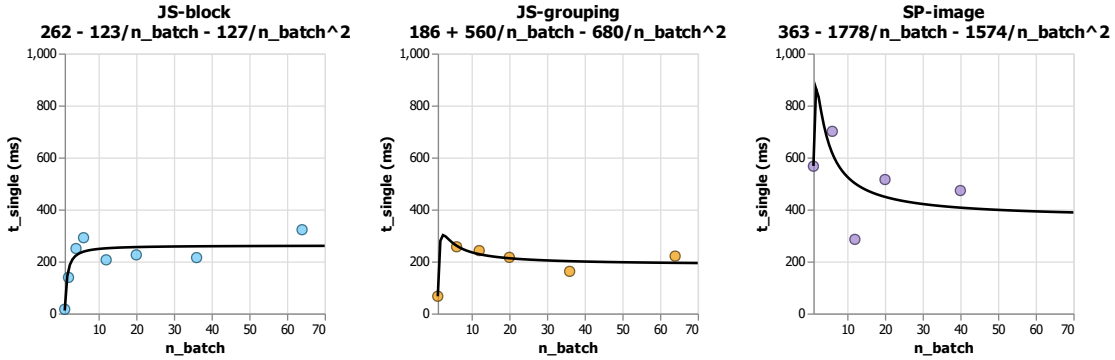


Figure C.5: The fitted t_{single} as a function of n_{batch} for JS-block, JS-grouping, and SP-image using $y = a + b/x + c/x^2$.

- **For $\mathbf{y = a + b/x + c/x^2}$**: The fitting result for t_{single} is shown in Fig. C.5. For SP-image, due to the noisy data, the fitted model contains a spike. The functions fitted are:
  - for JS-block:

$$t_{single} = 261.8083 - 123.2102/n_{batch} - 127.2516/n_{batch}^2$$
$$\text{with adjusted } R^2 = 0.7115, SE = 51.2691$$

  - for JS-grouping:

$$t_{single} = 186.0985 + 560.1921/n_{batch} - 680.1595/n_{batch}^2$$
$$\text{with adjusted } R^2 = 0.8363, SE = 28.4728$$

– for SP-image:

$$t_{single} = 363.5150 - 1778.1390/n_{batch} - 1574.2689/n_{batch}^2$$
with adjusted $R^2 = -0.3656, SE = 176.7216$

- **For y = a + blog$_2$(x + 1)**: The functions fitted are:

  – for JS-block:

  $$t_{single} = 80.7479 + 54.3526 log_2(n_{batch} + 1)$$
  with adjusted $R^2 = 0.3991, SE = 73.9862$

  – for JS-grouping:

  $$t_{single} = 117.6121 + 28.5219 log_2(n_{batch} + 1)$$
  with adjusted $R^2 = 0.0675, SE = 67.9599$

  – for SP-image:

  $$t_{single} = 628.0140 - 50.1844 log_2(n_{batch} + 1)$$
  with adjusted $R^2 = -0.1393, SE = 161.4152$

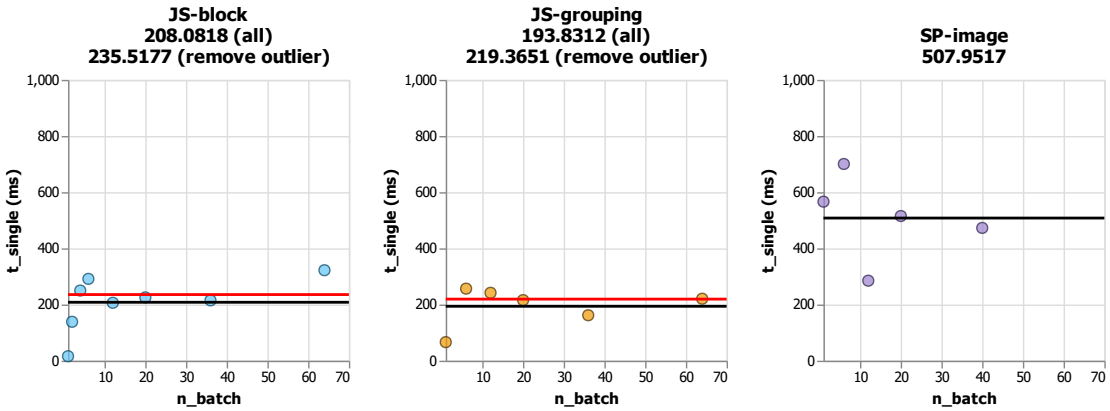


Figure C.6: The fitted t_{single} as a function of n_{batch} for JS-block, JS-grouping, and SP-image using $y = a$. The black line corresponds to the estimation without removing the outlier. The red line corresponds to the estimation after removing the outlier.

- **For y = a**: The fitting result for t_{single} is shown in Fig. C.6. The functions fitted are:

  – for JS-block:

  * If we use all the data points:

  $$t_{single} = 208.0818$$
  with adjusted $R^2 = 0, SE = 95.4480$

* If we remove the first data point, which seems an outlier:

$$t_{single} = 235.5177$$
$$\text{with adjusted } R^2 = 0, SE = 60.0269$$

– for JS-grouping:

* If we use all the data points:

$$t_{single} = 193.8312$$
$$\text{with adjusted } R^2 = 0, SE = 70.3764$$

* If we remove the first data point which seems an outlier:

$$t_{single} = 219.3651$$
$$\text{with adjusted } R^2 = 0, SE = 36.0720$$

– for SP-image:

$$t_{single} = 507.9517$$
$$\text{with adjusted } R^2 = 0, SE = 151.2289$$

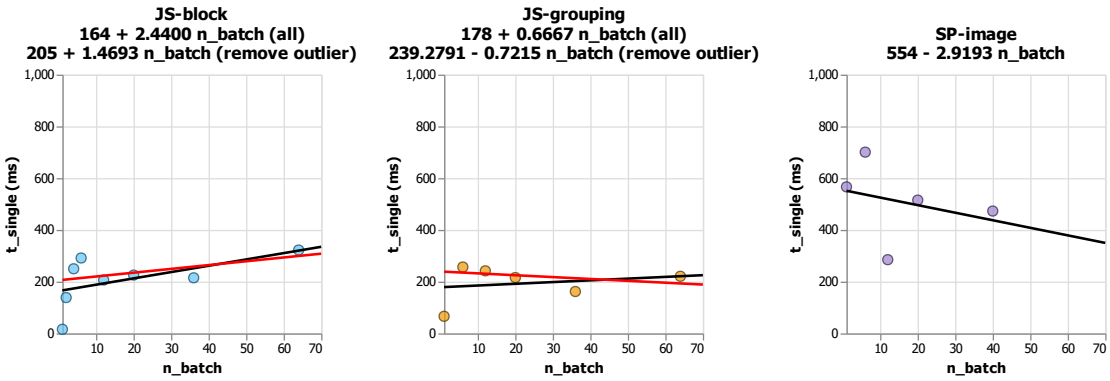


Figure C.7: The fitted t_{single} as a function of n_{batch} for JS-block, JS-grouping, and SP-image using $y = a + bx$. The black line corresponds to the estimation without removing the outlier. The red line corresponds to the estimation after removing the outlier.

- **For y = a + bx**: The fitting result for t_{single} is shown in Fig. C.7. The functions fitted are:

– for JS-block:

* If we use all the data points:

$$t_{single} = 163.8567 + 2.4400 n_{batch}$$
$$\text{with adjusted } R^2 = 0.1998, SE = 85.3806$$

    * If we remove the first data point, which seems an outlier:

$$t_{single} = 205.2919 + 1.4693 n_{batch}$$
$$\text{with adjusted } R^2 = 0.1630, SE = 54.9157$$

  - for JS-grouping:
    * If we use all the data points:

$$t_{single} = 178.3868 + 0.6667 n_{batch}$$
$$\text{with adjusted } R^2 = -0.1882, SE = 76.7149$$

    * If we remove the first data point, which seems an outlier:

$$t_{single} = 239.2791 - 0.7215 n_{batch}$$
$$\text{with adjusted } R^2 = -0.0448, SE = 36.8718$$

  - for SP-image

$$t_{single} = 554.0763 - 2.9193 n_{batch}$$
$$\text{with adjusted } R^2 = -0.2175, SE = 166.8644$$

Comments For JS-block and edges, $y = a + b/x + c/x^2$ is the best-fitting model among the three. For SP-image, all three models perform poorly because of the noisy and sparse samples.

For $y = a$, JS-block data fits okay with 1×1 and 8×8 being two significant outliers. As explained above, for 1×1, t_{single} approaches 0 because single edit and view are more concurrent than in other layouts. For 8×8, the large t_{single} is likely due to the latency of the interface. JS-grouping data fits well with 1×1 being the major outlier, and the reason is the same as for JS-block. SP-image data fits badly. 2×3 and 3×4 are two major outliers. However, note that the deviation at these two points roughly offsets, and thus there is still a possibility that $y = a$ suits the data.

For $y = a + bx$, for all three applications, the slopes are not too steep, and thus the analysis is similar to that of $y = a$.

Outcome We finally decide to use the simple constant function $y = a$ because:

- It makes sense logically. When there are more grid cells, the distance between grid cells decreases. The reduced distance makes it easier to conduct single edit. Meanwhile, the size of grid cells gets smaller, which makes it hard to single edit. These two effects may offset.
- Although $y = a + b/x + c/x^2$ numerically fits JS-block and edges very well, it fits SP-image poorly and thus may not be reliable.

- It is the simplest model and may avoid overfitting.

Besides, we decide not to discard the outlying points, as they do not change the resulting model much. The functions we choose are:

- **for JS-block: t_{single} = 208.0818**

  with adjusted $R^2 = 0, SE = 95.4480$

- **for JS-grouping: t_{single} = 193.8312**

  with adjusted $R^2 = 0, SE = 70.3764$

- **for SP-image: t_{single} = 507.9517**

  with adjusted $R^2 = 0, SE = 151.2289$

C.3.4 Reestimate t_{view} by t_{new} and t_{single} Models

Goal Reestimate t_{view} by removing the previous modeled t_{new} and t_{single} from the equations to possibly reduce the noise and make sure the original equations still approximately hold.

Procedure With the t_{new} and t_{single} modeled in the last sections, we reestimate t_{view} from the experiment data by removing the contribution of t_{new} and t_{single} to equations as

$$\left[t_{view}\right] = (X^T X)^{-1} X^T \begin{bmatrix} T_1 - N_{new,1}\, t_{new} - N_{single,1}\, t_{single} \\ T_2 - N_{new,2}\, t_{new} - N_{single,2}\, t_{single} \\ \cdots \\ T_n - N_{new,n}\, t_{new} - N_{single,n}\, t_{single} \end{bmatrix} \tag{C.3}$$

where X is the N_{view} measured for the n trials

$$X = \begin{bmatrix} N_{view,1} \\ N_{view,2} \\ \cdots \\ N_{view,n} \end{bmatrix}$$

and the t_{new} and t_{single} in the formula use modeled t_{new} and t_{single} values in the last section.

Outcome For t_{view}, the influence of the reestimation is shown in Fig. C.8. The first row is the initial estimations by multiple linear regression. The second row is after reestimation by putting t_{new} back. The third row is after reestimation by putting t_{new} and t_{single} back.

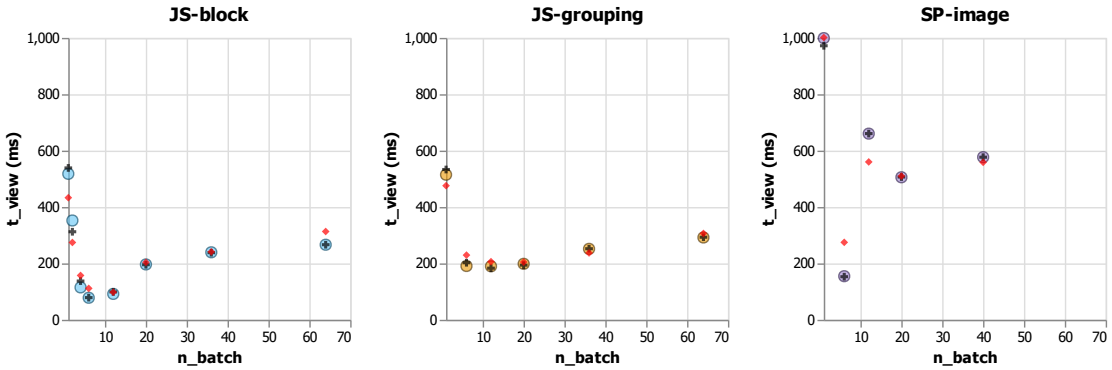


Figure C.8: t_{view} reestimation by t_{new} and t_{single}: The influence of reestimation to t_{view} as a function of n_{batch} for the three applications. The colored dots are original estimations by multiple linear regression. The black crosses are reestimations by putting only t_{new} back. The red diamonds are reestimations by putting both t_{new} and t_{single} back.

Comments The modeling of t_{new} does not significantly change the estimation of t_{view}. By comparison, the additional modeling of t_{single} changes t_{view} observably.

For JS-block, the 1×1 point lowers significantly after reestimation for t_{new} and t_{single} because the modeled t_{single} at 1×1 is much larger than the initial t_{single} which is close to 0. The t_{view} at 1×1 is decreased by the reestimation to compensate this change.

Similarly, for JS-grouping, the 1×1 point lowers significantly. For SP-image, the 2×3 point increases significantly while the 3×4 point decreases significantly.

All the other data points are changed little.

C.3.5 Model Reestimated t_{view}

Goal Model the reestimated t_{view} to reduce noise.

Procedure We model t_{view} as a function of n_{batch}. We fit a model for each application because we expect viewing time to be related to the content. For JS-block, there are 8 data points. For JS-grouping, there are 6 data points. For SP-image, there are 5 data points. We assume the suitable model function family should be shared among applications. We used the Online Nonlinear Regression Tool [282] to produce a set of candidate model functions $t_{new} = f(n_{batch})$.

- With the number of parameters = 2, we get the top 3 functions being:
  - for JS-block:
    - $y = ax^{b/x}$ with $R^2 = 0.5548$
    - $y = \frac{ax}{x+b}$ with $R^2 = 0.5397$
    - $y = ae^{b/x}$ with $R^2 = 0.5098$
  - for JS-grouping:

    * $y = \frac{ax}{x+b}$ with $R^2 = 0.8494$
    * $y = ae^{b/x}$ with $R^2 = 0.8477$
    * $y = a + b/x$ with $R^2 = 0.8174$
  - for SP-image:
    * $y = ax^{b/x}$ with $R^2 = 0.8815$
    * $y = \frac{ax}{x+b}$ with $R^2 = 0.7500$
    * $y = ae^{b/x}$ with $R^2 = 0.7289$

- With the number of parameters = 3, we get the top 3 functions being:
  - for JS-block:
    * $y = a + bx + c/x$ with $R^2 = 0.9310$
    * $y = x^a e^{bx^c}$ with $R^2 = 0.9093$
    * $y = a + bx + c/x^2$ with $R^2 = 0.8822$
  - for JS-grouping:
    * $y = a + bx + c/x$ with $R^2 = 0.9958$
    * $y = x^a e^{bx^c}$ with $R^2 = 0.9886$
    * $y = ax^b e^{cx}$ with $R^2 = 0.9874$
  - for SP-image:
    * $y = a + \frac{b}{x+c}$ with $R^2 = 0.9803$
    * $y = ae^{\frac{b}{x+c}}$ with $R^2 = 0.9795$
    * $y = a + b/x + c/x^2$ with $R^2 = 0.9546$

- With the number of parameters = 4, we get the top 3 functions being:
  - for JS-block:
    * $y = a + bx + c/x + dln(x)$ with $R^2 = 0.9378$
    * $y = a + bx + c/x + d/x^2$ with $R^2 = 0.9311$
    * $y = a + bx + c/x$ with $R^2 = 0.9310$
  - for JS-grouping:
    * $y = a + bx + c/x + dln(x)$ with $R^2 = 0.9998$
    * $y = a + bx + c/x + d/x^2$ with $R^2 = 0.9997$
    * $y = ae^{b/x} + ce^{d/x}$ with $R^2 = 0.9993$
  - for SP-image:
    * $y = \frac{a+bx+cx^2}{x+d}$ with $R^2 = 0.9862$

* $y = a + \frac{b}{x+c}$ with $R^2 = 0.9803$
* $y = ae^{\frac{b}{x+c}}$ with $R^2 = 0.9795$

It can be seen that $y = a + bx + c/x$ is reoccurring. It is the best for JS-block and JS-grouping when parameters = 3 and the third-best for JS-block when parameters = 4. Adding $dln(x)$ and d/x^2 terms generate the best and second best function for JS-block and JS-grouping when parameters = 4. Removing the cx term generates the third-best for JS-grouping when parameters = 2. For SP-image, the data is sparse and noisy and thus not decisive.

Thus, we choose $y = a + bx + c/x$ to be the function family for t_{view}. We fit the model with linear regression.

Table C.4: Modeled operator time costs: The final estimations of operator time costs.

Application	**Layout**	$\mathbf{t_{new}}$ (ms)	$\mathbf{t_{view}}$ (ms)	$\mathbf{t_{single}}$ (ms)
JS-block	1 × 1	443	438	208
JS-block	1 × 2	515	251	208
JS-block	2 × 2	691	164	208
JS-block	2 × 3	769	140	208
JS-block	3 × 4	859	134	208
JS-block	4 × 5	897	156	208
JS-block	6 × 6	924	217	208
JS-block	8 × 8	939	333	208
JS-grouping	1 × 1	443	477	194
JS-grouping	2 × 3	769	219	194
JS-grouping	3 × 4	859	206	194
JS-grouping	4 × 5	897	213	194
JS-grouping	6 × 6	924	242	194
JS-grouping	8 × 8	939	302	194
SP-image	1 × 1	443	988	508
SP-image	2 × 3	769	433	508
SP-image	3 × 4	859	421	508
SP-image	4 × 5	897	461	508
SP-image	5 × 8	927	601	508

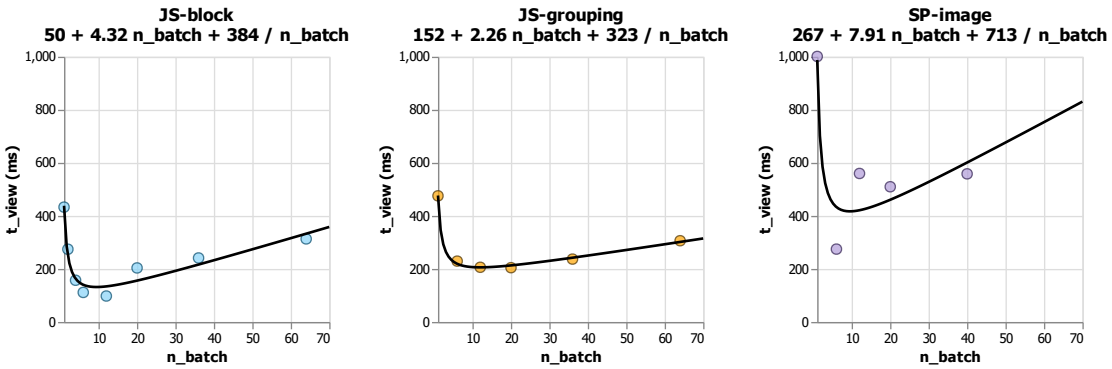


Figure C.9: The fitted t_{view} as a function of n_{batch} for the three applications using $y = a + bx + c/x$.

Outcome The fitting result for t_{view} is shown in Fig. C.9 The functions fitted are:

- **for JS-block:** $\mathbf{t_{view} = 50.4283 + 4.3202 n_{batch} + \frac{383.5033}{n_{batch}}}$

  with adjusted $R^2 = 0.9034, SE = 34.8005$

- **for JS-grouping:** $\mathbf{t_{view} = 151.9951 + 2.2618 n_{batch} + \frac{322.6178}{n_{batch}}}$

  with adjusted $R^2 = 0.9930, SE = 8.7040$

- **for SP-image:** $\mathbf{t_{view} = 267.0109 + 7.9103 n_{batch} + \frac{712.8504}{n_{batch}}}$

  with adjusted $R^2 = 0.6472, SE = 156.1362$

Comments The data clearly shows a U-shaped pattern. We conjecture that small grid cells require more attention and time to see clearly, while large grid cells can be distracting and require more eye movement to view one object.

$n_{batch} * gridCellArea = interfaceArea$ is a constant. If we assume the penalty of time cost to look into details of small objects to be inversely proportional to object area, the bx term can be explained. If we assume the basic eye moment time to scan an object to be proportional to the object's area, the c/x term can be explained. The view action clearly would contain some reaction latency of the human, and thus the a term can be explained.

C.4 Final Estimations of Operator Time Costs

Table C.4 shows the final estimations of operator time costs with the models we fitted for t_{new}, t_{view}, and t_{single}.

List of Figures

1.1 **What is data extraction**: Data extraction takes the bitmap image of a visualization as input and outputs the dataset that generates the visualization. For example, given a scanned image of Playfair's line chart on the left [210], the data table on the right shows the outcome of data extraction. 2

2.1 **Historical thematic maps**: (a) Edmond Halley's 1686 map showing directions of trade winds and monsoons [111]. (b) John Snow's 1855 map showing the distribution of cholera fatalities published in the book *"On the Mode of Communication of Cholera"* [250]. (c) Charles Joseph Minard's 1869 map showing successive losses of soldiers in Napoleon's Russian campaign [190]. (d) Charles Dupin's 1826 map showing the rate of male children in primary schools in each region [79]. 10

2.2 **Forerunners of common statistical charts**: (a) William Playfair's 1786 bar chart showing the distribution of exports and imports of Scotland [211]. (b) Nicole Oresme's fourteen-century illustration of functional relations resembling bar chart [202]. (c) William Playfair's 1786 line chart showing Britain's exports and imports to and from north America [211]. (d) A 10th-century plot showing planetary movement through zodiac [265]. (e) William Playfair's 1801 chart showing the population and revenues of several European countries [213]. (f) John Herschel's 1833 scatterplot for curve fitting [119]. 12

2.3 **Beyond the rectangular coordinate system**: (a) André-Michel Guerry's 1833 parallel coordinates showing crime frequencies in different age groups [107]. (b) Florence Nightingale's 1858 radar chart of causes of mortalities [198]. (c) Georg von Mayr's 1877 radar chart titled *"Linien-Diagramme im Kreise"* (translates to *"Line Charts in Circles"*) [184]. (d) William Farr's 1852 radial line chart showing mortalities in relation to temperature [86]. 14

2.4 **Joseph Priestley's timelines**: (a) Timeline titled *"A Chart of Biography"* published in 1765 showing lifespans of historical figures [217]. (b) Timeline titled *"A New Chart of History"* published in 1769 showing lifespans and territories of empires [218]. 15

2.5 **Historical ISOTYPE**: (a) Willard C. Brinton's 1914 ISOTYPE showing the railroad passenger growth [39]. (b) Otto Neurath's 1930 ISOTYPE showing potato production growth [193]. 16

3.1 The cholera map [250] shows the number of fatalities in the 1854 London Cholera Outbreak with a discrete bar chart. The fatalities are distributed at more than 300 locations on the map and visualized as stacked blocks. The right figure illustrates the visual encoding of the cholera map. A bar aggregates the fatalities at a location, where the number of blocks denotes the number of fatalities. A bar typically sits on a base with a root point denoting its location information. 46

3.2 Some examples of challenging cases in processing the cholera map image [250]: (a) A block conglutinates with the background; (b) A segment of the dashed line may be confused for a block; (c) The grouping of blocks is ambiguous; (d) Blocks are not distributed close to a baseline. 47

3.3 **The generic MI3 workflow**: The blue blocks denote algorithmic steps, and the green blocks denote interactive steps. The optional data preprocessing step transforms the raw data from the original problem into a classification task for a set of unlabeled data objects $\mathbf{D}_0$. We consider the $k+1$-th iteration of the active learning as an instance. Here, by "active learning", we refer to its general definition that the system actively seeks decisions from the user to aid the ML process. The algorithmic sampling step selects the data objects $\mathbf{S}_k$ to be labeled. The algorithmic default labeling step assigns default label annotation $\mathbf{A}_k$ for data objects $\mathbf{S}_k$. In the interactive labeling step, the user corrects the mislabeled data objects in the interface and produces ground truth (GT) labels $\mathbf{A}'_k$. The confirmed label annotations $\mathbf{A}'_k$ are then added to the labeled dataset. An interim classifier is trained with the partially labeled dataset, which can be used to compute default labels in the next iteration. The stoppage analysis step checks whether the interactive classification stage should stop. If not, the next session of interactive classification starts. Else, the interim classifier can be applied to label the remaining unlabeled data objects $\mathbf{D}_{k+1}$ with labels $\mathbf{B}_{k+1}$. The user goes through a quality assurance step to verify the labels for these data objects. The data objects $\mathbf{D}_{k+1}$ and verified labels $\mathbf{B}'_{k+1}$ are then added to the labeled dataset and form the labels $\mathbf{L}_0$ for the whole data object set $\mathbf{D}_0$. An optional postprocessing step transforms the labeled data objects into the desired final output data structure. The MI3 workflow is generic, and its components can be implemented with application-specific algorithms. 49

3.4 A data extraction procedure for spatial data visualization with three subroutines: block detection, block grouping, and root detection. Each of the three subtasks is an interactive classification task and is solved with an instantiation of the generic MI3 workflow. **(1) Block Detection**: Given an input image, we use connected components detection as a preprocessing to convert block detection into a classification task for connected components. **(2) Block Grouping**: With the detected blocks, we build a graph of block relations and classify the existence of relations to determine which pairs of blocks should be grouped in the same bar. The confirmed relations are used to compile the detection of bars. **(3) Root Detection**: For each bar, we generate candidate root points. Therefore, the task of determining the key point is transformed into the classification of candidate roots. 54

3.5 Candidate block detection process on part of the cholera map [250]: (a) Input part of the cholera map; (b) Binarize the image to reduce color variations; (c) Erode the image to reduce conglutination; (d) Detect connected components. 55

3.6 Measurements related to appearance features of candidate blocks. The left shows the size features and the angle. The right shows area definitions involved in the computation of shape features. 56

3.7 Neighborhood features and measurements. Horizontal/Vertical neighborhood refers to the union of 10 pixels horizontally/vertically extending outward the candidate block along its horizontal/vertical axis. The measurements of central horizontal and vertical neighborhoods are similar. Surrounding neighborhood refers to the union of pixels within 5 pixels' distance from the border of the candidate block. 56

3.8 Candidate neighboring relation detection result on part of the cholera map [250]. The blocks are denoted with green contours. The blue dots show the center of each block. The detected candidate neighboring relations are shown as cyan lines linking the centers of the two blocks involved. 57

3.9 Candidate root detection result on part of the cholera map [250]. The bars are denoted with green contours. Each candidate root is denoted with a cyan dot. 58

3.10 The first row shows the process of detecting alignment lines, which is used for computing the distances from candidate roots to alignment lines. The second row shows the process of detecting the skeletons, which is used for computing the distances from candidate roots to skeletons. . . 59

3.11 The prototype user interface for interactive classification and active learning. The interface consists of five parts: (a) Control Toolbar: the user can upload the image, select the chart type, switch interface layout, and change the algorithm parameters; (b) Image View: the Image View shows the input image and the re-visualized extracted dataset, and the user can navigate by pan and zoom, clip the image, set the scale of the image, manually create visual objects if needed, and conduct spatial filtering of data objects; (c) Annotation Panel: the user can annotate or correct the labels for data objects; (d) Dataset View: the Dataset View shows the extracted dataset as a table after the extraction process is finished; (e) Image Overview: the Image Overview highlights the currently zoomed region shown in the Image View. 63

3.12 Comparison of the 6 optional solutions to the block detection problem. The curves on the left show the change of interaction cost to achieve 100% accuracy with regard to the interaction cost in active learning. The bar chart on the right compares the performance of the optional solutions at 35, 70, 105, and 140 user operations for active learning. The correspondence between solutions and colored curves and bars is: blue for the brute force approach (**BF**); orange for the conventional ML-based approach with 20% data objects prelabeled and negative default labeling (**ML + 20%-PL + NeD + 6-DL**); green for random sampling with random default labeling (**RS + RD + 6-DL**); red for algorithmic sampling with random default labeling (**AS + RD + 6-DL**); purple for random sampling with algorithmic default labeling (**RS + AD + 6-DL**); brown for algorithmic sampling with algorithmic default labeling (**AS + AD + 6-DL**). All the solutions except **BF** show 6 data objects in each labeling session, and therefore, the "**6-DL**" is omitted in the legend. 69

3.13 Comparison of the 6 optional solutions in neighboring relation detection and root detection. The curves show the change of overall interaction costs with regard to the interaction cost in active learning. 71

3.14 Examples of historical visualizations: (a) Charles Minard's movement flow map, 1869 [190]; (b) William Playfair's bar chart, 1822 [214]; (c) Émile Cheysson's thematic map with bar-chart glyphs, 1906 [59]; (d) Charles Minard's bubble map, 1859 [189]; (e) Jacques Bertillon's thematic map with pie-chart glyphs, 1886 [27]; (f) Arthur Cayley's node-link diagram, 1890 [50]; (g) William Playfair's line chart, 1784 [209]; (h) William Phillips's scatterplot, 1958 [208]; (i) Adolphe d'Angeville's choropleth map, 1836 [70]. 73

3.15 The interface of the prototype line chart data extraction system. (a) **Start Point Detection**: The user clicks on the canvas to specify a coarse region of the start point and chooses the exact start point among the listed options. (b) **Next Point Tracking**: The user chooses the next point among candidates suggested by the algorithm. 79

3.16 The interface of the prototype Sankey diagram data extraction system. . 80

3.17 **Historical charts used by participants**: (a) *"Haejwa chŏndo"* published in 1822 showing eight provinces of Korea at the time [264]. The red lines show the admin boundaries of the provinces. The colored dots represent the counties. The counties in the same province share the same color. The black lines connecting counties represent the traffic routes. (b) A 1841 map of Xinhui County recorded in the county's local chronicles [173]. The map was created by Xingzhang Lin, the magistrate of Xinhui County, and Peifang Huang. The landmarks and villages are noted with text. The triangular marks show the distribution of military camps. The double rectangle marks show the distribution of forts. (c) Qian Sima's *"Yearly Chronicle of the Six States"* showing major historical events in the Warring States period of China. The timeline was first published in the book *"Shiji"* around 91 BCE [248]. The timeline reads from right to left. Each column corresponds to a year between 476 BCE and 207 BCE. Each row corresponds to a state. The figure comes from an edition by Shanfu Huang published in 1198 and shows the first two pages of the timeline. 82

4.1 Common APIs in data labeling systems. 100

4.2 OneLabeler's visual programming interface: (A) Workflow canvas panel for a developer to create and compose nodes, as well as editing the visual layout of the workflow; (B) Node configuration panel for configuring parameters of a selected node; (C) Console panel for showing static checking result of whether the workflow denotes a valid labeling system; (D) The variable inspector panel showing state values; (E) The generated data labeling system by OneLabeler according to the workflow. 103

4.3 A classification system for the customized webpage data type. 114

4.4 Building a machine-aided multi-task text labeling system: (A) User-configured workflow for the system. The interface of generated multi-task text labeling system: (B_1) A labeling panel showing the sentence for the annotator to conduct **interactive labeling**; (B_2) A list of created named entities; (B_3) A list of created entity relations. 115

4.5 Building a mixed-initiative image classification system: (A) User-configured workflow for the system. The interface of the generated image classification system: (B_1) A projection of data object features allowing annotators to use lasso selection for **data object selection**; (B_2) A labeling panel showing images for an annotator to conduct **interactive labeling**. 118

4.6 Building an interactive machine learning system for visualization data extraction: (A) User-configured workflow for the system. The interface of the generated system for visualization data extraction: (B_1) An overview image that allows the annotator to conduct **interactive labeling** and overview the data extraction result; (B_2) A labeling panel showing the geometric objects for the annotator to conduct **interactive labeling**. . . 119

4.7 A gallery of labeling systems built with OneLabeler using a single workflow (shown in Ⓐ) but different configurations for the interactive labeling module. The configurations differ in data type, label task type, and interface design. The labeling systems ⒸⒹⒺ all support classification, multi-label classification, and freeform text annotation, while additionally supporting label task types that are specialized to the corresponding data type (e.g., object detection for image). (A) A minimal data labeling template that can be used to generate the six labeling systems ⒷⒸⒹⒺⒻⒼ, with some modifications to node configurations for each labeling system; (B) An image classification system; (C) A multi-task image labeling system (additionally support object detection, and segmentation); (D) A multi-task text labeling system (additionally support span tagging, and span relation annotation); (E) A multi-task video labeling system (additionally support span tagging); (F) An audio freeform text annotation system; (G) A point cloud classification system. 121

5.1 **ML workflows in data-centric applications requiring quality assurance**: (a) A workflow to generate summary statistics from the classification of whether an image contains solar panels. Quality assurance of classification labels is needed to achieve reliable statistics. Following the terminology in Fig. 3.3 and Fig. 4.1, the step of "Applying an ML model" is a default labeling module, and "QA" refers to a quality assurance module. (b) A workflow with two stages (b_1 and b_2) to extract datasets from visualization images. Quality assurance is necessary to avoid propagating errors to the succeeding stage and guarantee the correctness of the extracted dataset. This workflow is a redraw of Fig. 3.3 where all the nodes in the active learning loop are wrapped into "Labeling with Active Learning". 132

5.2 **Examples of QA interfaces**: The interface components (editing panel, header toolbar, and context panel) are annotated with colored rectangles. The buttons corresponding to operations (new batch, single edit, batch edit) available to the user are annotated with colored ovals. The interfaces all contain an edit panel that presents a matrix of thumbnails showing data objects, a header toolbar providing interaction functions, and a context panel that provides an overview of the dataset: (a) An interface for solar panel detection corresponding to workflow in Fig. 5.1(a). (b) An interface for visual object detection corresponding to workflow in Fig. 5.1(b_1). (c) An interface for visual object grouping corresponding to workflow in Fig. 5.1(b_2). (d) The typical design of grid-based interfaces in QA. 133

5.3 A general workflow of user and machine routine operations in a quality assurance session in a grid-based interface typified in Fig. 5.2(d). The operations in the workflow can be categorized into five categories: user interaction in the interface, the user's viewing action, algorithmic operation, user decision, and algorithmic decision. 135

5.4 A workflow of human and machine operations in a grid-based interface without batch edit functions. This workflow simplifies the workflow in Fig. 5.3 by removing operations related to batch edit. 141

5.5 Different layouts of a grid-based interface for visual object detection in JS-block dataset: (a) 4×5 layout, (b) 1×1 layout, (c) 2×3 layout, (d) 3×4 layout, and (e) 6×6 layout. 142

5.6 Summary of the experiment result and fitted models: (a) Estimation of t_{new} by multiple linear regression and model of t_{new}. The selected model is $y = a + b/x + c/x^2$. (b) Initial estimations of t_{view} by multiple linear regression. (c) Initial estimations of t_{single} by multiple linear regression. (d - f) Models of t_{single} for the 3 cases. The selected model is $y = a$. The data points are t_{single} reestimated by putting t_{new} back. (g - i) Models of t_{view} for the 3 cases. The selected model is $y = a + bx + c/x$. The data points are t_{view} reestimated by putting t_{new} and t_{single} back. The x-axes of the figures are the number of grid cells n_{batch}. The y-axes of the figures are the corresponding operator time costs t_{new}, t_{view}, or t_{single}. The unit of all the time costs is milliseconds. 144

5.7 **Simulation of layouts**: The effect of grid layout in the three applications. For all the applications, the total time cost follows a U-shaped curve. For JS-block, the minimal is achieved at 4×5. For JS-grouping, the minimal is achieved at 4×5. For SP-image, the minimal is achieved at 3×4. . . . 148

5.8 **Variations of label strategy**: The simulation result of total time cost $T_{session}$ with 5 different label strategies: NoBC, OPT, Random, TOP2, TOP3. The starting point of the vertical axis is 200 instead of 0 to highlight the differences in vertical positions. 154

5.9 **Variations of default label accuracy**: The simulation result of total time cost $T_{session}$ with 5 conditions of default label accuracy, including 0.5, 0.6, 0.7, 0.8, and 0.9. 156

5.10 **Variations of rank method precision**: 5 conditions of rank method precision are simulated, including 0, 0.3, 0.56, 0.8, and 1. An additional condition called "no rank method" is simulated where the dataset is randomly ordered. The curves of "no rank method" and rank methods with precision 0 and 0.3 heavily overlap. The starting point of the vertical axis is 200 instead of 0 to highlight the differences in vertical positions. . 159

C.1 **Initial estimations of operator time costs**: The estimated operator time costs t_{new}, t_{view}, and t_{single} in relation to the grid layouts for all the applications. The error bars show the estimated standard error of the estimation. It can be seen that as the number of grid cells grows, t_{new} increases, t_{view} first increases then decreases, t_{single} first increases then stays stable. 184

C.2 The estimated model of t_{new} as a function of n_{batch} for the three applications using $y = a + b/x + c/x^2$. 187

C.3 $\mathbf{t_{view}}$ **reestimation by** $\mathbf{t_{new}}$: The influence of reestimation to t_{view} as a function of n_{batch} for the three applications. The colored dots are original estimations by multiple linear regression. The black crosses are reestimations. 188

C.4 $\mathbf{t_{single}}$ **reestimation by** $\mathbf{t_{new}}$: The influence of reestimation to t_{single} as a function of n_{batch} for the three applications. The colored dots are original estimations by multiple linear regression. The black crosses are reestimations. 188

C.5 The fitted t_{single} as a function of n_{batch} for JS-block, JS-grouping, and SP-image using $y = a + b/x + c/x^2$. 191

C.6 The fitted t_{single} as a function of n_{batch} for JS-block, JS-grouping, and SP-image using $y = a$. The black line corresponds to the estimation without removing the outlier. The red line corresponds to the estimation after removing the outlier. 192

C.7 The fitted t_{single} as a function of n_{batch} for JS-block, JS-grouping, and SP-image using $y = a + bx$. The black line corresponds to the estimation without removing the outlier. The red line corresponds to the estimation after removing the outlier. 193

C.8 **t_{view} reestimation by t_{new} and t_{single}**: The influence of reestimation to t_{view} as a function of n_{batch} for the three applications. The colored dots are original estimations by multiple linear regression. The black crosses are reestimations by putting only t_{new} back. The red diamonds are reestimations by putting both t_{new} and t_{single} back. 196
C.9 The fitted t_{view} as a function of n_{batch} for the three applications using $y = a + bx + c/x$. 199

List of Tables

2.1 The historical visualizations introduced in this section. The italicized name refers to an excerpt of the original title given by the visualization's author. 9

3.1 Optional pipelines for extracting data in Fig. 3.14(a - d), which may be configured using the 18 active learning workflows described earlier in this section. Each pipeline is denoted as $\mathbf{W}_{\text{datatype},k}$, where k indicates the k-th option. When a workflow is given a superscript "m", "a", or "ma", it is likely most cost-effective for the workflow to activate a purely manual, a simple algorithmic, or a parameterized algorithmic process instead of active learning. Almost all pipelines will require the workflow c_8 (measurement transformation), and thus it is not shown explicitly in the table. 76

3.2 Optional pipelines for extracting data in Fig. 3.14(e - i), which may be configured using the 18 active learning workflows described earlier in this section. See Table 3.1 for notational information. 77

4.1 Occurrences of states in the literature. The final codes are abbreviated by initials (i.e., "DO" for "data objects", "L" for "labels", "S" for "samples", "M" for "model", and "F" for "features"). Slash (/) is used in final codes when the flowchart contains no states or the flowchart's states are excluded in the coding procedure. 97

4.2 Occurrences of modules in the literature. The final codes are abbreviated by initials (i.e., "IL" for "interactive labeling", "DOS" for "data object selection", "MT" for "model training", "FE" for "feature extraction", "DL" for "default labeling", "QA" for "quality assurance", "SA" for "stoppage analysis", and "LI" for "label ideation"). 98

4.3 Coding the flowchart in AILA [64]. The final codes are abbreviated by initials. A slash (/) in the final code refers to the case that the theme is excluded from the final code. 98

4.4 Summary of task completion status and time (in minutes). 122

5.1 The parameters of our simulation model. Common mathematical notations are used to denote the data types, such as $\mathbb{N}$ for integers, $\mathbb{R}$ for real numbers, + for > 0, 0+ for ≥ 0, $[a, b]$ for a real range. 160

C.1 **Experiment design**: The number of repeated trials conducted for each combination of the independent variables: application, grid layout, number of data objects. 181

C.2 **Experiment design continued**: The number of repeated trials conducted for each combination of the independent variables: application, grid layout, number of data objects. 182

C.3 **Initial estimations of operator time costs**: The estimated time costs of unit user operations using multiple linear regression. For each estimation, the estimated standard error is shown in the parenthesis. The statistical significance is marked with ** meaning $p \leq 0.01$ and * meaning $p \leq 0.05$. R^2 columns show the R-squared of the entire linear model $T_{round} = N_{new}t_{new} + N_{view}t_{view} + N_{single}t_{single}$. It can be seen that for all the experimented applications and grid layouts, N_{new} contributes linearly to T_{round}. In most cases, N_{view} and N_{single} contributes linearly to T_{round}. 185

C.4 **Modeled operator time costs**: The final estimations of operator time costs. 198

List of Algorithms

1 Transition Matrix Computation 62
2 Incremental Label Propagation 62
3 QA Session Simulation 138
4 Grid-Based Interface (Single Edit) 148

References

[1] Martín Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*. OSDI ’16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283.

[2] Alfie Abdul-Rahman, Gleen Roe, Mark Olsen, Clovis Gladstone, Richard Whaling, Nicholas Cronk, Robert Morrissey, and Min Chen. “Constructive Visual Analytics for Text Similarity Detection”. In: *Computer Graphics Forum* 36.1 (2017), pp. 237–248.

[3] Johnny Accot and Shumin Zhai. “Beyond Fitts’ Law: Models for Trajectory-Based HCI Tasks”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’97. New York, NY, USA: ACM, 1997, pp. 295–302.

[4] Luis von Ahn and Laura Dabbish. “Labeling Images with a Computer Game”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’04. Vienna, Austria: ACM, 2004, pp. 319–326.

[5] Luis von Ahn and Laura Dabbish. “Designing Games with a Purpose”. In: *Communications of the ACM* 51.8 (Aug. 2008), pp. 58–67.

[6] Luis von Ahn, Ruoran Liu, and Manuel Blum. “Peekaboom: A Game for Locating Objects in Images”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’06. Montreal, QC, Canada: ACM, 2006, pp. 55–64.

[7] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. “reCAPTCHA: Human-Based Character Recognition via Web Security Measures”. In: *Science* 321.5895 (Aug. 2008), pp. 1465–1468.

[8] James Frederick Allen, Curry I. Guinn, and Eric Horvitz. “Mixed-initiative interaction”. In: *IEEE Intelligent Systems and their Applications* 14.5 (Sept. 1999), pp. 14–23.

[9] Bilal Alsallakh, Allan Hanbury, Helwig Hauser, Silvia Miksch, and Andreas Rauber. “Visual Methods for Analyzing Probabilistic Classification Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 1703–1712.

[10] Amazon. *Amazon Mechanical Turk*. 2005.

[11] Saleema Amershi. “Designing for Effective End-User Interaction with Machine Learning”. PhD thesis. University of Washington, 2012.

[12] John R. Anderson. *The Architecture of Cognition*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 1983. 345 pp.

[13] John R. Anderson. *Rules of the Mind*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 1993.

[14] Keith Andrews. “Evaluation Comes in Many Guises”. In: *Proceedings of the CHI Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization*. BELIV ’08. Florence, Italy, Apr. 2008, pp. 7–8.

[15] Mykhaylo Andriluka, Jasper R. R. Uijlings, and Vittorio Ferrari. “Fluid Annotation: A Human-Machine Collaboration Interface for Full Image Annotation”. In: *Proceedings of the ACM International Conference on Multimedia*. MM ’18. Seoul, Republic of Korea: ACM, 2018, pp. 1957–1966.

[16] Mihael Ankerst, Christian Elsen, Martin Ester, and Hans-Peter Kriegel. “Visual Classification: An Interactive Approach to Decision Tree Construction”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’99. San Diego, California, USA: ACM, 1999, pp. 392–396.

[17] Nathaniel Ayewah, William Pugh, David Hovemeyer, J. David Morgenthaler, and John Penix. “Using Static Analysis to Find Bugs”. In: *IEEE Software* 25.5 (Sept. 2008), pp. 22–29.

[18] Leif Azzopardi. “The Economics in Interactive Information Retrieval”. In: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’11. Beijing, China: ACM, 2011, pp. 15–24.

[19] Leif Azzopardi, Diane Kelly, and Kathy Brennan. “How Query Cost Affects Search Behavior”. In: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’13. Dublin, Ireland: ACM, 2013, pp. 23–32.

[20] Louise Barkhuus and Jennifer A. Rode. “From Mice to Men - 24 Years of Evaluation in CHI”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’07. San Jose, California, USA: ACM, 2007.

[21] Leilani Battle, Peitong Duan, Zachery Miranda, Dana Mukusheva, Remco Chang, and Michael Stonebraker. “Beagle: Automated Extraction and Interpretation of Visualizations from the Web”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’18. New York, NY, USA: ACM, Nov. 2018, pp. 1–8.

[22] A. Bäuerle, H. Neumann, and T. Ropinski. “Classifier-Guided Visual Correction of Noisy Labels for Image Classification Tasks”. In: *Computer Graphics Forum* 39.3 (June 2020), pp. 195–205.

[23] David V. Beard, Dana K. Smith, and Kevin M. Denelsbeck. “Quick and Dirty GOMS: A Case Study of Computed Tomography Interpretation”. In: *Human-Computer Interaction* 11.2 (June 1996), pp. 157–180.

[24] Jürgen Bernard, Marco Hutter, Matthias Zeppelzauer, Dieter Fellner, and Michael Sedlmair. “Comparing Visual-Interactive Labeling with Active Learning: An Experimental Study”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 298–308.

[25] Jürgen Bernard, Christian Ritter, David Sessler, Matthias Zeppelzauer, Jörn Kohlhammer, and Dieter Fellner. “Visual-Interactive Similarity Search for Complex Objects by Example of Soccer Player Analysis”. In: *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. INSTICC. Setúbal, Portugal: SciTePress, 2017, pp. 75–87.

[26] Jürgen Bernard, Matthias Zeppelzauer, Michael Sedlmair, and Wolfgang Aigner. “VIAL: a unified process for visual interactive labeling”. In: *The Visual Computer* 34.9 (2018), pp. 1189–1207.

[27] Jacques Bertillon. *Paris 1886 Population par Groupes d’Âges et par Quartier*. Paris, France, 1886.

[28] BIOSOFT. *UnGraph*. 2004.

[29] Michael Bloodgood and K. Vijay-Shanker. “A Method for Stopping Active Learning Based on Stabilizing Predictions and the Need for User-Adjustable Stopping”. In: *Proceedings of the Conference on Computational Natural Language Learning*. CoNLL ’09. Boulder, Colorado: Association for Computational Linguistics, 2009, pp. 39–47.

[30] Avrim Blum and Tom Mitchell. “Combining Labeled and Unlabeled Data with Co-Training”. In: *Proceedings of the Annual Conference on Computational Learning Theory*. COLT ’98. Madison, WI, USA: ACM, 1998, pp. 92–100.

[31] Maryam Booshehrian, Torsten Möller, Randall M. Peterman, and Tamara Munzner. “Vismon: Facilitating Analysis of Trade-Offs, Uncertainty, and Sensitivity In Fisheries Management Decision Making”. In: *Computer Graphics Forum* 31.3 (June 2012), pp. 1235–1244.

[32] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. “D3: Data-Driven Documents”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (Nov. 2011), pp. 2301–2309.

[33] N. Boukhelifa, A. Bezerianos, R. Chang, C. Collins, S. Drucker, A. Endert, J. Hullman, C. North, M. Sedlmair, and Theresa-Marie Rhyne. “Challenges in Evaluating Interactive Visual Machine Learning Systems”. In: *IEEE Computer Graphics and Applications* 40.6 (2020), pp. 88–96.

[34] Simon Bovet and Jean Bovet. *GraphClick*. 2006.

[35] Yuri Boykov, Olga Veksler, and Ramin Zabih. “Fast Approximate Energy Minimization via Graph Cuts”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.11 (2001), pp. 1222–1239.

[36] Steve Branson, Catherine Wah, Florian Schroff, Boris Babenko, Peter Welinder, Pietro Perona, and Serge Belongie. “Visual Recognition with Humans in the Loop”. In: *Proceedings of the European Conference on Computer Vision*. ECCV ’10. Berlin, Heidelberg: Springer, 2010, pp. 438–451.

[37] Virginia Braun, Victoria Clarke, Nikki Hayfield, and Gareth Terry. “Thematic Analysis”. In: *Handbook of Research Methods in Health Social Sciences*. Singapore: Springer, 2019, pp. 843–860.

[38] Klaus Brinker. “Incorporating Diversity in Active Learning with Support Vector Machines”. In: *Proceedings of the International Conference on Machine Learning*. ICML ’03. Washington, DC, USA: AAAI Press, 2003, pp. 59–66.

[39] Willard Cope Brinton. *Graphic Methods for Presenting Facts*. New York, NY, USA: Engineering Magazine Company, 1914. 371 pp.

[40] Carla E. Brodley and Mark A. Friedl. “Identifying Mislabeled Training Data”. In: *Journal of Artificial Intelligence Research* 11.1 (July 1999), pp. 131–167.

[41] John Brosz, Miguel A. Nacenta, Richard Pusch, Sheelagh Carpendale, and Christophe Hurter. “Transmogrification: Causal Manipulation of Visualizations”. In: *Proceedings of the ACM Symposium on User Interface Software and Technology*. UIST ’13. St. Andrews, Scotland, UK: ACM, 2013, pp. 97–106.

[42] Nicholas J. Bryan, Gautham J. Mysore, and Ge Wang. “ISSE: An Interactive Source Separation Editor”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. Toronto, ON, Canada: ACM, 2014, pp. 257–266.

[43] Maya Cakmak and Andrea L. Thomaz. “Mixed-Initiative Active Learning”. In: *Proceedings of ICML Workshop on Combining Learning Strategies to Reduce Label Cost*. 2011.

[44] Francisco Maria Calisto, Carlos Santiago, Nuno Nunes, and Jacinto C. Nascimento. “Introduction of human-centric AI assistant to aid radiologists for multimodal breast image classification”. In: *International Journal of Human-Computer Studies* 150 (2021), p. 102607.

[45] Sandra Carberry, Stephanie Elzer, and Seniz Demir. “Information Graphics: An Untapped Resource for Digital Libraries”. In: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’06. New York, NY, USA: ACM, 2006, pp. 581–588.

[46] Jaime Carbonell. “AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction”. In: *IEEE Transactions on Man Machine Systems* 11.4 (Dec. 1970), pp. 190–202.

[47] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, eds. *Readings in Information Visualization: Using Vision to Think*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[48] Stuart K. Card, Thomas P. Moran, and Allen Newell. “The Keystroke-Level Model for User Performance Time with Interactive Systems”. In: *Communications of the ACM* 23.7 (July 1980), pp. 396–410.

[49] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 1983. 469 pp.

[50] Arthur Cayley. *The Collected Mathematical Papers of Arthur Cayley, Volume 3*. Cambridge, UK: Cambridge University Press, 1890.

[51] Chengliang Chai, Guoliang Li, Ju Fan, and Yuyu Luo. “Crowdsourcing-based Data Extraction from Visualization Charts”. In: *Proceedings of the IEEE International Conference on Data Engineering*. ICDE ’20. Dallas, TX, USA: IEEE, 2020, pp. 1814–1817.

[52] Joseph Chee Chang, Saleema Amershi, and Ece Kamar. “Revolt: Collaborative Crowdsourcing for Labeling Machine Learning Datasets”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, CO, USA: ACM, May 2017, pp. 2334–2346.

[53] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. “Cluster Kernels for Semi-Supervised Learning”. In: *Proceedings of the International Conference on Neural Information Processing Systems*. NIPS ’02. Vancouver, BC, Canada: MIT Press, 2002, pp. 601–608.

[54] Duen Horng Chau, Aniket Kittur, Jason I. Hong, and Christos Faloutsos. “Apolo: Making Sense of Large Network Data by Combining Rich User Interaction and Machine Learning”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’11. Vancouver, BC, Canada: ACM, 2011, pp. 167–176.

[55] Changjian Chen, Zhaowei Wang, Jing Wu, Xiting Wang, Lan-Zhe Guo, Yu-Feng Li, and Shixia Liu. “Interactive Graph Construction for Graph-Based Semi-Supervised Learning”. In: *IEEE Transactions on Visualization and Computer Graphics* 27 (9 2021), pp. 3701–3716.

[56] Quan Ze Chen, Tobias Schnabel, Besmira Nushi, and Saleema Amershi. "HINT: Integration Testing for AI-Based Features with Humans in the Loop". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '22. Helsinki, Finland: ACM, 2022, pp. 549–565.

[57] Zhe Chen, Michael Cafarella, and Eytan Adar. "DiagramFlyer: A Search Engine for Data-Driven Diagrams". In: *Proceedings of the International Conference on World Wide Web*. WWW '15 Companion. New York, NY, USA: ACM, 2015, pp. 183–186.

[58] Justin Cheng and Michael S. Bernstein. "Flock: Hybrid Crowd-Machine Learning Classifiers". In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work & Social Computing*. CSCW '15. Vancouver, BC, Canada: ACM, Feb. 2015, pp. 600–611.

[59] Émile Cheysson. *Circulation sur les Routes Nationales, de 1851 à 1894*. Paris, France, 1906.

[60] Ed Huai-Hsin Chi. "A Taxonomy of Visualization Techniques using the Data State Reference Model". In: *Proceedings of the IEEE Symposium on Information Visualization*. INFOVIS '00. Salt Lake City, UT, USA: IEEE, Oct. 2000, pp. 69–75.

[61] Ed Huai-Hsin Chi and John Thomas Riedl. "An Operator Interaction Framework for Visualization Systems". In: *Proceedings of the IEEE Symposium on Information Visualization*. INFOVIS '98. Research Triangle Park, NC, USA: IEEE, Oct. 1998, pp. 63–70.

[62] Yao-Yi Chiang, Stefan Leyk, and Craig A. Knoblock. "A Survey of Digital Map Processing Techniques". In: *ACM Computing Surveys* 47.1 (May 2014), pp. 1–44.

[63] Jinho Choi, Sanghun Jung, Deok Gun Park, Jaegul Choo, and Niklas Elmqvist. "Visualizing for the Non-Visual: Enabling the Visually Impaired to Use Visualization". In: *Computer Graphics Forum* 38.3 (2019), pp. 249–260.

[64] Minsuk Choi, Cheonbok Park, Soyoung Yang, Yonggyu Kim, Jaegul Choo, and Sungsoo Ray Hong. "AILA: Attentive Interactive Labeling Assistant for Document Classification through Attention-Based Deep Neural Networks". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '19. New York, NY, USA: ACM, May 2019, pp. 1–12.

[65] Gerald P. Chubb. "SAINT, A Digital Simulation Language for the Study of Manned Systems". In: *Manned Systems Design*. Ed. by Jan Moraal and Karl-Friedrich Kraiss. New York, NY, USA: Plenum Press, 1981, pp. 153–179.

[66] William S. Cleveland and Robert McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods". In: *Journal of the American Statistical Association* 79.387 (Sept. 1984), pp. 531–554.

[67] Jingyu Cui, Fang Wen, Rong Xiao, Yuandong Tian, and Xiaoou Tang. "EasyAlbum: An Interactive Photo Annotation System Based on Face Clustering and Re-ranking". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, CA, USA: ACM, 2007, pp. 367–376.

[68] Weiwei Cui, Xiaoyu Zhang, Yun Wang, He Huang, Bei Chen, Lei Fang, Haidong Zhang, Jian-Guan Lou, and Dongmei Zhang. "Text-to-Viz: Automatic Generation of Infographics from Proportion-Related Natural Language Statements". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020), pp. 906–916.

[69] Adolphe d'Angeville. *Essai Sur La Statistique De La Population Française, Considérée Sous Quelques-uns De Ses Rapports Physiques Et Moraux*. Bourg, Gironde, France: Fred Dufour, 1836.

[70] Adolphe d'Angeville. *Statistique de la Population Française, Population Habitans par myriametres carries en 1831*. Bourg, Gironde, France, 1836.

[71] Filip Dabek and Jesus J. Caban. "A Grammar-based Approach for Modeling User Interactions and Generating Suggestions During the Data Exploration Process". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 41–50.

[72] Jia Deng, Olga Russakovsky, Jonathan Krause, Michael Bernstein, Alex Berg, and Li Fei-Fei. "Scalable Multi-Label Annotation". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14. Toronto, ON, Canada: ACM, 2014, pp. 3099–3102.

[73] René Descartes. *La Géométrie*. 1637. 91 pp.

[74] Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. "Solving the multiple instance problem with axis-parallel rectangles". In: *Artificial Intelligence* 89.1 (Jan. 1997), pp. 31–71.

[75] Alan Dix, Janet Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction*. London, UK: Pearson Education, 2004. 834 pp.

[76] Rusty Dodson and Waldo Tobler. *Snow's Cholera Map*. 1994.

[77] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017.

[78] John J. Dudley and Per Ola Kristensson. "A Review of User Interface Design for Interactive Machine Learning". In: *ACM Transactions on Interactive Intelligent Systems* 8.2 (June 2018).

[79] Charles Dupin. *Carte figurative de l'instruction populaire de la France*. Paris, 1826.

[80] Ernest Edmonds. "The Future of Intelligent Interfaces: Not Just "How?", but "What?" and "Why?"" In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '93. Orlando, FL, USA: ACM, 1993, pp. 13–17.

[81] Joscha Eirich, Jakob Bonart, Dominik Jackle, Michael Sedlmair, Ute Schmid, Kai Fischbach, Tobias Schreck, and Jurgen Bernard. "IRVINE: A Design Study on Analyzing Correlation Patterns of Electrical Engines". In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2021), pp. 11–21.

[82] Sara Evensen, Chang Ge, and Cagatay Demiralp. "Ruler: Data Programming by Demonstration for Document Labeling". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1996–2005.

[83] Explosion. *Prodigy*. 2017.

[84] Jerry Alan Fails and Dan R. Olsen. "Interactive Machine Learning". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '03. Miami, FL, USA: ACM, 2003, pp. 39–45.

[85] Arlen Fan, Yuxin Ma, Michelle Mancenido, and Ross Maciejewski. "Annotating Line Charts for Addressing Deception". In: *SIGCHI Conference on Human Factors in Computing Systems*. CHI '22. New Orleans, LA, USA: ACM, 2022.

[86] William Farr. *Temperature and Mortality of London, for every week of 11 years (1840-50)*. 1852.

[87] Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S. Bernstein. "Iris: A Conversational Agent for Complex Tasks". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal, QC, Canada: ACM, 2018, pp. 1–12.

[88] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. "From Data Mining to Knowledge Discovery in Databases". In: *AI Magazine* 17.3 (1996), pp. 37–54.

[89] Cristian Felix, Aritra Dasgupta, and Enrico Bertini. "The Exploratory Labeling Assistant: Mixed-Initiative Label Curation with Large Document Collections". In: *Proceedings of the ACM Symposium on User Interface Software and Technology*. UIST '18. Berlin, Germany: ACM, 2018, pp. 153–164.

[90] Paul M. Fitts. "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement". In: *Journal of Experimental Psychology: General* 121.3 (1992), pp. 262–269.

[91] James Fogarty, Desney Tan, Ashish Kapoor, and Simon Winder. "CueFlik: Interactive Concept Learning in Image Search". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 29–38.

[92] Jules Françoise, Baptiste Caramiaux, and Téo Sanchez. "Marcelle: Composing Interactive Machine Learning Workflows and Interfaces". In: *Proceedings of the ACM Symposium on User Interface Software and Technology*. UIST '21. Virtual Event, USA: ACM, 2021, pp. 39–53.

[93] Michael Friendly. "The Golden Age of Statistical Graphics". In: *Statistical Science* 23.4 (2008), pp. 502–535.

[94] Michael Friendly and Daniel J. Denis. *Milestones in the history of thematic cartography, statistical graphics, and data visualization*. 2001.

[95] Michael Friendly, Pedro Valero-Mora, and Joaquín Ibáñez Ulargui. "The First (Known) Statistical Graph: Michael Florent van Langren and the "Secret" of Longitude". In: *The American Statistician* 64.2 (May 2010), pp. 174–184.

[96] Michael Friendly and Howard Wainer. *A History of Data Visualization and Graphic Communication*. Cambridge, MA, USA: Harvard University Press, 2021. 308 pp.

[97] Howard Gray Funkhouser. "A Note on a Tenth Century Graph". In: *Osiris* 1 (1936), pp. 260–262.

[98] Krzysztof Gajos and Daniel S. Weld. "SUPPLE: Automatically Generating User Interfaces". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '04. Funchal, Madeira, Portugal: ACM, 2004, pp. 93–100.

[99] P. Gamba and A. Mecocci. "Perceptual Grouping for Symbol Chain Tracking in Digitized Topographic Maps". In: *Pattern Recognition Letters* 20.4 (Apr. 1999), pp. 355–365.

[100] Jinglun Gao, Yin Zhou, and Kenneth E. Barner. "View: Visual Information Extraction Widget for Improving Chart Images Accessibility". In: *Proceedings of the IEEE International Conference on Image Processing*. ICIP '12. Orlando, FL, USA: IEEE, Sept. 2012, pp. 2865–2868.

[101] Humberto S. Garcia Caballero, Michel A. Westenberg, Binyam Gebre, and Jarke J. van Wijk. "V-Awake: A Visual Analytics Approach for Correcting Sleep Predictions from Deep Learning Models". In: *Computer Graphics Forum* 38.3 (2019), pp. 1–12.

[102] Melinda T. Gervasio, Michael D. Moffitt, Martha E. Pollack, Joseph M. Taylor, and Tomas E. Uribe. "Active Preference Learning for Personalized Calendar Scheduling Assistance". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '05. San Diego, CA, USA: ACM, 2005, pp. 90–97.

[103] Richard Gong and David Kieras. "A Validation of the GOMS Model Methodology in the Development of a Specialized, Commercial Software Application". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '94. Boston, MA, USA: ACM, 1994, pp. 351–357.

[104] Wayne D. Gray and Marilyn C. Salzman. "Damaged Merchandise? A Review of Experiments That Compare Usability Evaluation Methods". In: *Human–Computer Interaction* 13.3 (Sept. 1998), pp. 203–261.

[105] Spence Green, Jeffrey Heer, and Christopher D. Manning. "The Efficacy of Human Post-Editing for Language Translation". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, 2013, pp. 439–448.

[106] Arnd Gross, Sibylle Schirm, and Markus Scholz. "Ycasd - a tool for capturing and scaling data from graphical representations". In: *BMC Bioinformatics* 15.1 (2014), p. 219.

[107] André-Michel Guerry. *Essai sur la statistique morale de la France*. Paris, France: Crochard, 1833. 69 pp.

[108] Yves Guiard, Michel Beaudouin-Lafon, Julien Bastin, Dennis Pasveer, and Shumin Zhai. "View Size and Pointing Difficulty in Multi-Scale Navigation". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. New York, NY, USA: ACM, 2004, pp. 117–124.

[109] Yuhong Guo and Dale Schuurmans. "Discriminative Batch Mode Active Learning". In: *Proceedings of the International Conference on Neural Information Processing Systems*. NIPS '07. Vancouver, BC, Canada: Curran Associates Inc., 2007, pp. 593–600.

[110] Zicheng Guo and Richard W. Hall. "Parallel Thinning with Two-Subiteration Algorithms". In: *Communications of the ACM* 32.3 (1989), pp. 359–373.

[111] Edmond Halley. "An Historical Account of the Trade Winds, and Monsoons, Observable in the Seas between and Near the Tropicks, with an Attempt to Assign the Phisical Cause of the Said Winds". In: *Philosophical Transactions of the Royal Society of London* 16.183 (Sept. 1686), pp. 153–168.

[112] Gaudenz Halter, Rafael Ballester-Ripoll, Barbara Flueckiger, and Renato Pajarola. "VIAN: A Visual Annotation Tool for Film Analysis". In: *Computer Graphics Forum* 38.3 (2019), pp. 119–129.

[113] Steve Haroz, Robert Kosara, and Steven L. Franconeri. "ISOTYPE Visualization: Working Memory, Performance, and Engagement with Pictographs". In: *Proceedings of the ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: ACM, 2015, pp. 1191–1200.

[114] Jonathan Harper and Maneesh Agrawala. “Deconstructing and Restyling D3 Visualizations”. In: *Proceedings of the ACM Symposium on User Interface Software and Technology*. UIST ’14. New York, NY, USA: ACM, 2014, pp. 253–262.

[115] Francis Hauksbee. “X. An account of an experiment touching the ascent of water between two glass planes, in an hyperbolick figure”. In: *Philosophical Transactions of the Royal Society of London* 27.336 (Dec. 1710), pp. 539–540.

[116] Jeffrey Heer and Michael Bostock. “Crowdsourcing Graphical Perception: Using Mechanical Turk to Assess Visualization Design”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’10. New York, NY, USA: ACM, 2010, pp. 203–212.

[117] William E. Hefley and Dianne Murray. “Intelligent User Interfaces”. In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI ’93. Orlando, FL, USA: ACM, 1993, pp. 3–10.

[118] Florian Heimerl, Steffen Koch, Harald Bosch, and Thomas Ertl. “Visual Classifier Training for Text Document Retrieval”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dec. 2012), pp. 2839–2848.

[119] John Herschel. “On the Investigation of the Orbits of revolving Double Stars”. In: *Memoirs of the Royal Astronomical Society* 5 (Jan. 1833), pp. 171–222.

[120] William Edmund Hick. “On the Rate of Gain of Information”. In: *The Quarterly Journal of Experimental Psychology* 4.1 (Mar. 1952), pp. 11–26.

[121] Benjamin Höferlin, Rudolf Netzel, Markus Höferlin, Daniel Weiskopf, and Gunther Heidemann. “Inter-Active Learning of Ad-Hoc Classifiers for Video Visual Analytics”. In: *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*. VAST ’12. Seattle, WA, USA: IEEE, Oct. 2012, pp. 23–32.

[122] Steven C. H. Hoi and Michael R. Lyu. “A Semi-Supervised Active Learning Framework for Image Retrieval”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. CVPR ’05. San Diego, CA, USA: IEEE, 2005, pp. 302–309.

[123] K. Höök. “Steps to take before intelligent user interfaces become real”. In: *Interacting with Computers* 12.4 (2000), pp. 409–426.

[124] Eric Horvitz. “Uncertainty, Action, and Interaction: In Pursuit of Mixed-Initiative Computing”. In: *IEEE Intelligent Systems* (1999), pp. 17–20.

[125] Mei-Ling Hsu. “The Qin maps: A clue to later Chinese cartographic development”. In: *Imago Mundi* 45.1 (1993), pp. 90–100.

[126] Xian-Sheng Hua and Guo-Jun Qi. “Online Multi-Label Active Annotation: Towards Large-Scale Content-Based Video Search”. In: *Proceedings of the ACM International Conference on Multimedia*. MM ’08. Vancouver, BC, Canada: ACM, 2008, pp. 141–150.

[127] Thomas S. Huang, Charlie K. Dagli, Shyamsundar Rajaram, Edward Y. Chang, Michael I. Mandel, Graham E. Poliner, and Daniel P. W. Ellis. “Active Learning for Interactive Multimedia Retrieval”. In: *Proceedings of the IEEE* 96.4 (Apr. 2008), pp. 648–667.

[128] Weihua Huang and Chew Lim Tan. “A System for Understanding Imaged Infographics and Its Applications”. In: *Proceedings of the ACM Symposium on Document Engineering*. DocEng ’07. Winnipeg, MB, Canada: ACM, 2007, pp. 9–18.

[129] Weihua Huang, Chew Lim Tan, and Wee Kheng Leow. "Elliptic Arc Vectorization for 3D Pie Chart Recognition". In: *Proceedings of the International Conference on Image Processing*. Vol. 5. IEEE, 2004, pp. 2889–2892.

[130] Alfred Inselberg. "The plane with parallel coordinates". In: *The Visual Computer* 1.2 (Aug. 1985), pp. 69–91.

[131] Liu Jiang, Shixia Liu, and Changjian Chen. "Recent research advances on interactive machine learning". In: *Journal of Visualization* 22.2 (Nov. 2019), pp. 401–417.

[132] Thorsten Joachims. "Transductive Inference for Text Classification Using Support Vector Machines". In: *Proceedings of the International Conference on Machine Learning*. ICML '99. Bled, Slovenia: Morgan Kaufmann Publishers Inc., 1999, pp. 200–209.

[133] Bonnie E. John and David E. Kieras. "The GOMS family of user interface analysis techniques: comparison and contrast". In: *ACM Transactions on Computer-Human Interaction* 3.4 (Dec. 1996), pp. 320–351.

[134] Bonnie E. John and David E. Kieras. "Using GOMS for User Interface Design and Evaluation: Which Technique?" In: *ACM Transactions on Computer-Human Interaction* 3.4 (Dec. 1996), pp. 287–319.

[135] Bonnie E. John and Alonso H. Vera. "A GOMS Analysis of a Graphic Machine-Paced, Highly Interactive Task". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '92. Monterey, California, USA: ACM, 1992, pp. 251–258.

[136] Bonnie Elizabeth John. "Contributions to Engineering Models of Human-computer Interaction". PhD thesis. USA: Carnegie Mellon University, 1988.

[137] Eric J. Johnson and Daniel Goldstein. "Do Defaults Save Lives?" In: *Science* 302.5649 (Nov. 2003), pp. 1338–1339.

[138] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. "ChartSense: Interactive Data Extraction from Chart Images". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, CO, USA: ACM, 2017, pp. 6706–6717.

[139] Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Akos Kadar, Adam Trischler, and Yoshua Bengio. "FigureQA: An Annotated Figure Dataset for Visual Reasoning". In: (2017). arXiv: `1710.07300 [cs.CV]`.

[140] Richard M. Karp. "A Note on the Application of Graph Theory to Digital Computer Programming". In: *Information and Control* 3.2 (June 1960), pp. 179–190.

[141] Saurabh Kataria, William Browuer, Prasenjit Mitra, and Clyde Lee Giles. "Automatic Extraction of Data Points and Text Blocks from 2-dimensional Plots in Digital Documents". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI '08. Chicago, IL, USA: AAAI Press, 2008, pp. 1169–1174.

[142] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. "Visual Analytics: Definition, Process, and Challenges". In: *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer, 2008, pp. 154–175.

[143] Caitlin Kelleher and Randy Pausch. "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers". In: *ACM Computing Surveys* 37.2 (June 2005), pp. 83–137.

[144] Alex B. W. Kennedy and H. Riall Sankey. “The Thermal Efficiency of Steam Engines”. In: *Minutes of the Proceedings of the Institution of Civil Engineers* 134.1898 (Jan. 1898), pp. 278–312.

[145] Norman Kerle and Jan de Leeuw. “Reviving Legacy Population Maps With Object-Oriented Image Processing Techniques”. In: *IEEE Transactions on Geoscience and Remote Sensing* 47.7 (July 2009), pp. 2392–2402.

[146] Roli Khanna, Jonathan Dodge, Andrew Anderson, Rupika Dikkala, Jed Irvine, Zeyad Shureih, Kin-Ho Lam, Caleb R. Matthews, Zhengxian Lin, Minsuk Kahng, Alan Fern, and Margaret Burnett. “Finding AI’s Faults with AAR/AI: An Empirical Study”. In: *ACM Transactions on Interactive Intelligent Systems* 12.1 (Mar. 2022).

[147] David E. Kieras. “Towards a Practical GOMS Model Methodology for User Interface Design”. In: *Handbook of Human-Computer Interaction*. Ed. by Martin G. Helander. Amsterdam, The Netherlands: Elsevier, 1988, pp. 135–157.

[148] David E. Kieras and Anthony J. Hornof. “Towards Accurate and Practical Predictive Models of Active-Vision-Based Visual Search”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. Toronto, ON, Canada: ACM, 2014, pp. 3875–3884.

[149] David E. Kieras and Thomas P. Santoro. “Computational GOMS Modeling of a Complex Team Task: Lessons Learned”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’04. Vienna, Austria: ACM, 2004, pp. 97–104.

[150] David E. Kieras, Scott D. Wood, and David E. Meyer. “Predictive Engineering Models Based on the EPIC Architecture for a Multimodal High-Performance Human-Computer Interaction Task”. In: *ACM Transactions on Computer-Human Interaction* 4.3 (Sept. 1997), pp. 230–275.

[151] Klokan Technologies GmbH. *Old Maps Online*.

[152] Tom Koch. “The Map as Intent: Variations on the Theme of John Snow”. In: *Cartographica* 39.4 (Jan. 2004), pp. 1–14.

[153] Nicholas Kong and Maneesh Agrawala. “Graphical Overlays: Using Layered Elements to Aid Chart Reading”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dec. 2012), pp. 2631–2638.

[154] Nicholas Kong, Marti A. Hearst, and Maneesh Agrawala. “Extracting References Between Text and Charts via Crowdsourcing”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. Toronto, ON, Canada: ACM, 2014, pp. 31–40.

[155] Karl-Friedrich Kraiss. “A Display Design and Evaluation Study Using Task Network Models”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.5 (May 1981), pp. 339–351.

[156] Kostiantyn Kucher, Carita Paradis, Magnus Sahlgren, and Andreas Kerren. “Active Learning and Visual Analytics for Stance Classification with ALVA”. In: *ACM Transactions on Interactive Intelligent Systems* 7.3 (Oct. 2017).

[157] Giovane R. Kuhn, Manuel M. Oliveira, and Leandro A. F. Fernandes. “An Efficient Naturalness-Preserving Image-Recoloring Method for Dichromats”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (Nov. 2008), pp. 1747–1754.

[158] Todd Kulesza, Saleema Amershi, Rich Caruana, Danyel Fisher, and Denis Charles. "Structured Labeling for Facilitating Concept Evolution in Machine Learning". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14. Toronto, ON, Canada: ACM, 2014, pp. 3075–3084.

[159] Branislav Kveton and Shlomo Berkovsky. "Minimal Interaction Content Discovery in Recommender Systems". In: *ACM Transactions on Interactive Intelligent Systems* 6.2 (2016).

[160] Michael van Langren. *La Verdadera Longitud por Mar y Tierra*. Antwerp, Belgium, 1644. 13 pp.

[161] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. "Evaluation Strategies for HCI Toolkit Research". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal, QC, Canada: ACM, 2018, pp. 1–17.

[162] Po-shen Lee, Jevin D. West, and Bill Howe. "Viziometrics: Analyzing Visual Information in the Scientific Literature". In: *IEEE Transactions on Big Data* 4.1 (Mar. 2018), pp. 117–129.

[163] Christian Leistner, Amir Saffari, Jakob Santner, and Horst Bischof. "Semi-Supervised Random Forests". In: *Proceedings of the IEEE International Conference on Computer Vision*. ICCV '09. Kyoto, Japan: IEEE, Sept. 2009, pp. 506–513.

[164] Fritz Lekschas, Brant Peterson, Daniel Haehn, Eric Ma, Nils Gehlenborg, and Hanspeter Pfister. "Peax: Interactive Visual Pattern Search in Sequential Data Using Unsupervised Deep Representation Learning". In: *Computer Graphics Forum* 39.3 (June 2020), pp. 167–179.

[165] Clayton Lewis, Peter G. Polson, Cathleen Wharton, and John Rieman. "Testing a Walkthrough Methodology for Theory-Based Design of Walk-up-and-Use Interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. Seattle, WA, USA: ACM, 1990, pp. 235–242.

[166] Clayton Lewis and John Rieman. *Task-Centered User Interface Design*. 1993. 170 pp.

[167] David D. Lewis and Jason Catlett. "Heterogeneous Uncertainty Sampling for Supervised Learning". In: *Proceedings of the International Conference on Machine Learning*. ICML '94. New Brunswick, NJ, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 148–156.

[168] Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J. Franklin. "Crowdsourced Data Management: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering* 28.9 (Sept. 2016), pp. 2296–2319.

[169] Hongsen Liao, Li Chen, Yibo Song, and Hao Ming. "Visualization-Based Active Learning for Video Annotation". In: *IEEE Transactions on Multimedia* 18.11 (Nov. 2016), pp. 2196–2205.

[170] Library of Congress. *Library of Congress Digital Collections*.

[171] Christopher H. Lin, Mausam, and Daniel S. Weld. "Re-Active Learning: Active Learning with Relabeling". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI '16. Phoenix, AZ, USA: AAAI Press, 2016, pp. 1845–1852.

[172] Tzuta Lin. *LabelImg*. 2015.

[173] Xingzhang Lin and Peifang Huang. *Map of Xinhui County*. 1841.

[174] Dong Liu, Meng Wang, Xian-Sheng Hua, and Hong-Jiang Zhang. “Smart Batch Tagging of Photo Albums”. In: *Proceedings of the ACM International Conference on Multimedia*. MM ’09. Beijing, China: ACM, 2009, pp. 809–812.

[175] Ruizhe Liu, Weihua Huang, and Chew Lim Tan. “Extraction of Vectorized Graphical Information from Scientific Chart Images”. In: *Proceedings of the International Conference on Document Analysis and Recognition*. Vol. 1. ICDAR ’07. Parana, Brazil: IEEE, Sept. 2007, pp. 521–525.

[176] Shixia Liu, Changjian Chen, Yafeng Lu, Fangxin Ouyang, and Bin Wang. “An Interactive Method to Improve Crowdsourced Annotations”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), pp. 235–245.

[177] Min Lu, Jie Liang, Yu Zhang, Guozheng Li, Siming Chen, Zongru Li, and Xiaoru Yuan. “Interaction+: Interaction Enhancement for Web-based Visualizations”. In: *Proceedings of the IEEE Pacific Visualization Symposium*. IEEE, Apr. 2017, pp. 61–70.

[178] Yiwen Luo, Wei Liu, Jianzhuang Liu, and Xiaoou Tang. “MQSearch: Image Search by Multi-Class Query”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’08. Florence, Italy: ACM, 2008, pp. 49–52.

[179] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. “Learning Word Vectors for Sentiment Analysis”. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. HLT ’11. Portland, Oregon, USA: Association for Computational Linguistics, 2011, pp. 142–150.

[180] I. Scott MacKenzie and William Buxton. “Extending Fitts’ Law to Two-Dimensional Tasks”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’92. Monterey, CA, USA: ACM, 1992, pp. 219–226.

[181] Jock Mackinlay. “Automating the Design of Graphical Presentations of Relational Information”. In: *ACM Transactions on Graphics* 5.2 (Apr. 1986), pp. 110–141.

[182] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. “Building a Large Annotated Corpus of English: The Penn Treebank”. In: *Computational Linguistics* 19.2 (June 1993), pp. 313–330.

[183] Mark T. Maybury and Wolf Wahlster. “Intelligent User Interfaces: An Introduction”. In: *Readings in Intelligent User Interfaces*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 1–13.

[184] Georg von Mayr. *Die Gesetzmäßigkeit im Gesellschaftsleben*. Munich, German: Oldenbourg Wissenschaftsverlag, Dec. 1877.

[185] Gonzalo Gabriel Méndez, Miguel A. Nacenta, and Sebastien Vandenheste. “iVoLVER: Interactive Visual Language for Visualization Extraction and Reconstruction”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’16. San Jose, CA, USA: ACM, 2016, pp. 4073–4085.

[186] Qiguang Miao, Pengfei Xu, Xuelong Li, Jianfeng Song, Weisheng Li, and Yun Yang. “The Recognition of the Point Symbols in the Scanned Topographic Maps”. In: *IEEE Transactions on Image Processing* 26.6 (June 2017), pp. 2751–2766.

[187] Charles Joseph Minard. *Carte de la Circulation des Voyageurs par Voitures Publiques sur les routes de la contrée où sera placé le Chemin de Fer de Dijon à Mulhouse*. Paris, France, 1845.

[188] Charles Joseph Minard. *Tableaux Graphiques et des Cartes Figuratives*. Paris, France: E. Thunot, 1845.

[189] Charles Joseph Minard. *Carte Figurative et approximative des tonnages des Grand Ports et des principales Rivières d'Europe*. Paris, France, 1859.

[190] Charles Joseph Minard. *Carte figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813*. Paris, France, Nov. 1869.

[191] Rathin Radhakrishnan Nair, Nishant Sankaran, Ifeoma Nwogu, and Venu Govindaraju. "Automated Analysis of Line Plots in Documents". In: *Proceedings of the International Conference on Document Analysis and Recognition*. IEEE, Aug. 2015, pp. 796–800.

[192] Hiroki Nakayama, Takahiro Kubo, Junya Kamura, Yasufumi Taniguchi, and Xu Liang. *doccano: Text Annotation Tool for Human*. 2018.

[193] Otto Neurath. *Gesellschaft und Wirtschaft*. Leipzig, Germany: Bibliographisches Institut, 1930. 130 pp.

[194] Otto Neurath. *Empiricism and Sociology*. Dordrecht, The Netherlands: D. Reidel Publishing Company, 1973. 475 pp.

[195] Anh Nguyen and Wayne Wobcke. "An Agent-Based Approach to Dialogue Management in Personal Assistants". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '05. San Diego, California, USA: ACM, 2005, pp. 137–144.

[196] Hieu T. Nguyen and Arnold Smeulders. "Active Learning Using Pre-clustering". In: *Proceedings of the International Conference on Machine Learning*. ICML '04. Banff, AB, Canada: ACM, 2004, p. 79.

[197] Jakob Nielsen and Rolf Molich. "Heuristic Evaluation of User Interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. Seattle, WA, USA: ACM, 1990, pp. 249–256.

[198] Florence Nightingale. *Mortality of the British Army: At Home and Abroad, and During the Russian War, as Compared with the Mortality of the Civil Population in England*. London, UK: Harrison and Sons, 1858.

[199] Florence Nightingale. *Notes on Matters Affecting the Health, Efficiency, and Hospital Administration of the British Army*. London, UK: Harrison and Sons, 1858.

[200] Curtis Northcutt, Lu Jiang, and Isaac Chuang. "Confident Learning: Estimating Uncertainty in Dataset Labels". In: *Journal of Artificial Intelligence Research* 70 (May 2021), pp. 1373–1411.

[201] Allard Oelen, Markus Stocker, and Sören Auer. "Crowdsourcing Scholarly Discourse Annotations". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '21. College Station, TX, USA: ACM, 2021, pp. 464–474.

[202] Nicole Oresme. *Tractatus de latitudinibus formarum*. Padua, Italy: Mathaeus Cerdonis, 1482.

[203] Antti Oulasvirta, Per Ola Kristensson, Xiaojun Bi, and Andrew Howes, eds. *Computational Interaction*. Oxford, UK: Oxford University Press, Mar. 2018.

[204] Jose Gustavo S. Paiva, William Robson Schwartz, Helio Pedrini, and Rosane Minghim. "An Approach to Supporting Incremental Visual Data Classification". In: *IEEE Transactions on Visualization and Computer Graphics* 21.1 (Jan. 2015), pp. 4–17.

[205] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. “Librispeech: An ASR corpus based on public domain audio books”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. ICASSP ’15. South Brisbane, QLD, Australia: IEEE, 2015, pp. 5206–5210.

[206] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Proceedings of the International Conference on Neural Information Processing Systems*. NeuralIPS ’19. Red Hook, NY, USA: Curran Associates Inc., 2019.

[207] Aria Pezeshk and Richard L. Tutwiler. “Automatic Feature Extraction and Text Recognition From Scanned Topographic Maps”. In: *IEEE Transactions on Geoscience and Remote Sensing* 49.12 (Dec. 2011), pp. 5047–5063.

[208] Alban William Housego Phillips. “The Relation between Unemployment and the Rate of Change of Money Wage Rates in the United Kingdom, 1861-1957”. In: *Economica* 25.100 (1958), pp. 283–299.

[209] William Playfair. *Exports & Imports to and from all North-America*. London, UK, 1786.

[210] William Playfair. *Exports & Imports to and from Germany*. 1786.

[211] William Playfair. *The Commercial and Political Atlas: Representing, by Means of Stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure, and Debts of England, during the Whole of the Eighteenth Century*. London, UK: Debrett, Robinson, and Sewell, 1786.

[212] William Playfair. *Statistical Representation of the United States of America*. 1801.

[213] William Playfair. *The Statistical Breviary: Shewing, on a Principle Entirely New, the Resources of Every State and Kingdom in Europe*. London, UK: Wallis, 1801. 64 pp.

[214] William Playfair. *Chart Shewing at One View the Price of the Quarter of Wheat & Wages of Labour by the Week from the Year 1565 to 1821*. London, UK, 1822.

[215] Jorge Poco and Jeffrey Heer. “Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images”. In: *Computer Graphics Forum* 36.3 (June 2017), pp. 353–363.

[216] Jorge Poco, Angela Mayhua, and Jeffrey Heer. “Extracting and Retargeting Color Mappings from Bitmap Images of Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 637–646.

[217] Joseph Priestley. *A Chart of Biography*. London, 1765.

[218] Joseph Priestley. *A New Chart of History*. London, 1769.

[219] Claudius Ptolemy. *Clavdii Ptholemei Alexandrini philosophi Cosmographia*. Rome: Arnold Buckinck, 1478.

[220] Dimitrios Rafailidis, Apostolos Axenopoulos, Jonas Etzold, Stavroula Manolopoulou, and Petros Daras. “Content-Based Tag Propagation and Tensor Factorization for Personalized Item Recommendation Based on Social Tagging”. In: *ACM Transactions on Interactive Intelligent Systems* 3.4 (Jan. 2014).

[221] Anjana Ramkumar, Pieter Jan Stappers, Wiro J. Niessen, Sonja Adebahr, Tanja Schimek-Jasch, Ursula Nestle, and Yu Song. “Using GOMS and NASA-TLX to Evaluate Human-Computer Interaction Process in Interactive Segmentation”. In: *International Journal of Human-Computer Interaction* 33.2 (2017), pp. 123–134.

[222] Gonzalo Ramos, Christopher Meek, Patrice Simard, Jina Suh, and Soroush Ghorashi. "Interactive machine teaching: a human-centered approach to building machine-learned models". In: *Human–Computer Interaction* 35.5-6 (2020), pp. 413–451.

[223] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. "Snorkel: Rapid Training Data Creation with Weak Supervision". In: *Proceedings of the VLDB Endowment* 11.3 (Nov. 2017), pp. 269–282.

[224] Sagnik Ray Choudhury and Clyde Lee Giles. "An Architecture for Information Extraction from Figures in Digital Libraries". In: *Proceedings of the International Conference on World Wide Web*. WWW '15 Companion. New York, NY, USA: ACM, 2015, pp. 667–672.

[225] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. "Learning From Crowds". In: *The Journal of Machine Learning Research* 11 (Aug. 2010), pp. 1297–1322.

[226] Donghao Ren, Tobias Hollerer, and Xiaoru Yuan. "iVisDesigner: Expressive Interactive Design of Information Visualizations". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 2092–2101.

[227] Tim Rietz and Alexander Maedche. "Cody: An AI-Based System to Semi-Automate Coding for Qualitative Research". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '21. New York, NY, USA: ACM, 2021.

[228] John H. Rogers. "Origins of the ancient constellations: I. The Mesopotamian traditions". In: *Journal of the British Astronomical Associatio* 108.1 (1998), pp. 9–28.

[229] Ankit Rohatgi. *WebPlotDigitizer*. 2011.

[230] Ork de Rooij, Jarke J. van Wijk, and Marcel Worring. "MediaTable: Interactive Categorization of Multimedia Collections". In: *IEEE Computer Graphics and Applications* 30.5 (Sept. 2010), pp. 42–51.

[231] David Rumsey. *David Rumsey Map Collection*.

[232] Olga Russakovsky, Li-Jia Li, and Li Fei-Fei. "Best of both worlds: Human-machine collaboration for object annotation". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '15. Piscataway, NJ, USA: IEEE, June 2015, pp. 2121–2131.

[233] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. "LabelMe: A Database and Web-Based Tool for Image Annotation". In: *International Journal of Computer Vision* 77.1-3 (2008), pp. 157–173.

[234] Dominik Sacha, Matthias Kraus, Daniel A. Keim, and Min Chen. "VIS4ML: An Ontology for Visual Analytics Assisted Machine Learning". In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), pp. 385–395.

[235] Dominik Sacha, Michael Sedlmair, Leishi Zhang, John A. Lee, Jaakko Peltonen, Daniel Weiskopf, Stephen C. North, and Daniel A. Keim. "What you see is what you can change: Human-centered machine learning by interactive visualization". In: *Neurocomputing* 268 (Dec. 2017), pp. 164–175.

[236] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. "ReVision: Automated Classification, Analysis and Redesign of Chart Images". In: *Proceedings of the ACM Symposium on User Interface Software and Technology*. UIST '11. Santa Barbara, CA, USA: ACM, 2011, pp. 393–402.

[237] Christian Schüldt, Ivan Laptev, and Barbara Caputo. "Recognizing Human Actions: A Local SVM Approach". In: *Proceedings of the International Conference on Pattern Recognition*. Vol. 3. ICPR '04. Cambridge, UK: IEEE, 2004, pp. 32–36.

[238] Andrew Sears and Julie A. Jacko, eds. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Boca Raton, FL, USA: CRC Press, 2007. 1358 pp.

[239] Edward Segel and Jeffrey Heer. "Narrative Visualization: Telling Stories with Data". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (Nov. 2010), pp. 1139–1148.

[240] Christin Seifert and Michael Granitzer. "User-Based Active Learning". In: *IEEE International Conference on Data Mining Workshops*. Sydney, NSW, Australia: IEEE, Dec. 2010, pp. 418–425.

[241] Burr Settles. *Active Learning Literature Survey*. Technical Report. University of Wisconsin–Madison, 2009.

[242] Burr Settles. "Closing the Loop: Fast, Interactive Semi-supervised Annotation with Queries on Features and Instances". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP '11. Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pp. 1467–1478.

[243] Burr Settles, Mark Craven, and Lewis Friedland. "Active Learning with Real Annotation Costs". In: *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*. 2008, pp. 1–10.

[244] Xindi Shang, Donglin Di, Junbin Xiao, Yu Cao, Xun Yang, and Tat-Seng Chua. "Annotating Objects and Relations in User-Generated Videos". In: *Proceedings of the International Conference on Multimedia Retrieval*. ICMR '19. Ottawa, ON, Canada: ACM, 2019, pp. 279–287.

[245] Narushige Shiode, Shino Shiode, Elodie Rod-Thatcher, Sanjay Rana, and Peter Vinten-Johansen. "The mortality rates and the space-time patterns of John Snow's cholera epidemic map". In: *International Journal of Health Geographics* 14.1 (June 2015).

[246] Ben Shneiderman and Pattie Maes. "Direct Manipulation vs. Interface Agents". In: *Interactions* 4.6 (1997), pp. 42–61.

[247] Noah Siegel, Zachary Horvitz, Roie Levin, Santosh Divvala, and Ali Farhadi. "FigureSeer: Parsing Result-Figures in Research Papers". In: *Proceedings of the European Conference on Computer Vision*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. ECCV '16. Amsterdam, The Netherlands: Springer International Publishing, 2016, pp. 664–680.

[248] Qian Sima. *Shiji*. Shanfu, Huang, 1198.

[249] Qian Sima. *Yearly Chronicle of the Six States*. 1198.

[250] John Snow. *On the Mode of Communication of Cholera*. London, UK: John Churchill, 1855. 162 pp.

[251] Ove von Spaeth. “Dating the Oldest Egyptian Star Map”. In: *Centaurus* 42.3 (2000), pp. 159–179.

[252] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. “explAIner: A Visual Analytics Framework for Interactive and Explainable Machine Learning”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 1064–1074.

[253] Bongwon Suh and Benjamin B. Bederson. “Semi-Automatic Photo Annotation Strategies Using Event Based Clustering and Clothing Based Person Recognition”. In: *Interacting with Computers* 19.4 (July 2007), pp. 524–544.

[254] Qixiang Tan, ed. *The Historical Atlas of China*. Beijing, China: China Cartographic Publishing House, 1982.

[255] Jinhui Tang, Qiang Chen, Meng Wang, Shuicheng Yan, Tat-Seng Chua, and Ramesh Jain. “Towards Optimizing Human Labeling for Interactive Image Tagging”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 9.4 (Aug. 2013).

[256] James J. Thomas and Kristin A. Cook. *Illuminating the Path. The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Center, 2005. 184 pp.

[257] Yuandong Tian, Wei Liu, Rong Xiao, Fang Wen, and Xiaoou Tang. “A Face Annotation Framework with Partial Clustering and Interactive Labeling”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVPR ’07. Minneapolis, MN, USA: IEEE, June 2007, pp. 1–8.

[258] Maxim Tkachenko, Mikhail Malyuk, Nikita Shevchenko, Andrey Holmanyuk, and Nikolai Liubimov. *Label Studio: Data labeling software*. 2020.

[259] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. “Adapting User Interfaces with Model-Based Reinforcement Learning”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’21. Yokohama, Japan: ACM, 2021.

[260] Wei Tong and Rong Jin. “Semi-Supervised Learning by Mixed Label Propagation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI ’07. Vancouver, BC, Canada: AAAI Press, 2007, pp. 651–656.

[261] Edward Rolf Tufte. *The Visual Display of Quantitative Information*. Cheshire, CT, USA: Graphics Press, 1983. 197 pp.

[262] Bas Tummers. *DataThief III*. 2006.

[263] United States Census Bureau. *United States Census Bureau Publications*.

[264] unknown. *Haejwa chŏndo*. 1822.

[265] unknown. *Planetary Movements Shown as Cyclic Inclinations Over Time*. 901–1000.

[266] P. Utrilla, C. Mazo, M. C. Sopena, M. Martínez-Bea, and R. Domingo. “A palaeolithic map from 13,660 calBP: engraved stone blocks from the Late Magdalenian in Abauntz Cave (Navarra, Spain)”. In: *Journal of Human Evolution* 57.2 (2009), pp. 99–111.

[267] Sarah Theres Völkel, Christina Schneegass, Malin Eiband, and Daniel Buschek. “What is "Intelligent" in Intelligent User Interfaces? A Meta-Analysis of 25 Years of IUI”. In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI ’20. Cagliari, Italy: ACM, 2020, pp. 477–487.

[268] Guotai Wang, Maria A. Zuluaga, Wenqi Li, Rosalind Pratt, Premal A. Patel, Michael Aertsen, Tom Doel, Anna L. David, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. "DeepIGeoS: A Deep Interactive Geodesic Framework for Medical Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.7 (July 2019), pp. 1559–1572.

[269] Jiachen Wang, Kejian Zhao, Dazhen Deng, Anqi Cao, Xiao Xie, Zheng Zhou, Hui Zhang, and Yingcai Wu. "Tac-Simur: Tactic-based Simulative Visual Analytics of Table Tennis". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020), pp. 407–417.

[270] Meng Wang and Xian-Sheng Hua. "Active Learning in Multimedia Annotation and Retrieval: A Survey". In: *ACM Transactions on Intelligent Systems and Technology* 2.2 (Feb. 2011).

[271] Malcolm Ware, Eibe Frank, Geoffrey Holmes, Mark Hall, and Ian H. Witten. "Interactive machine learning: letting users build classifiers". In: *International Journal of Human-Computer Studies* 55.3 (2001), pp. 281–292.

[272] Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. "Machine Learning for User Modeling". In: *User Modeling and User-Adapted Interaction* 11.1-2 (2001), pp. 19–29.

[273] Julie S. Weber and Martha E. Pollack. "Entropy-Driven Online Active Learning for Interactive Calendar Management". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '07. Honolulu, HI, USA: ACM, 2007, pp. 141–150.

[274] Thomas Weber, Heinrich Hußmann, Zhiwei Han, Stefan Matthes, and Yuanting Liu. "Draw with Me: Human-in-the-Loop for Image Restoration". In: *Proceedings of the International Conference on Intelligent User Interfaces*. IUI '20. Cagliari, Italy: ACM, 2020, pp. 243–253.

[275] Jarke J. van Wijk. "The Value of Visualization". In: *Proceedings of the IEEE Conference on Visualization*. VIS '05. Minneapolis, MN, USA: IEEE, 2005, pp. 79–86.

[276] Leland Wilkinson. *The Grammar of Graphics*. Springer, June 30, 2005. 690 pp.

[277] Jiajun Wu, Yibiao Zhao, Jun-Yan Zhu, Siwei Luo, and Zhuowen Tu. "MILCut: A Sweeping Line Multiple Instance Learning Paradigm for Interactive Image Segmentation". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '14. IEEE, June 2014.

[278] Yingcai Wu, Ji Lan, Xinhuan Shu, Chenyang Ji, Kejian Zhao, Jiachen Wang, and Hui Zhang. "iTTVis: Interactive Visualization of Table Tennis Data". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 709–718.

[279] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '15. Boston, MA, USA: IEEE, 2015, pp. 1912–1920.

[280] Shouxing Xiang, Xi Ye, Jiazhi Xia, Jing Wu, Yang Chen, and Shixia Liu. "Interactive Correction of Mislabeled Training Data". In: *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*. VAST '19. Vancouver, BC, Canada: IEEE, 2019, pp. 57–68.

[281] Zuobing Xu, Ram Akella, and Yi Zhang. "Incorporating Diversity and Density in Active Learning for Relevance Feedback". In: *Proceedings of the European Conference on IR Research*. ECIR '07. Rome, Italy: Springer-Verlag, 2007, pp. 246–257.

[282] Xuru. *Online Regression Tools*.

[283] Hiromitsu Yamada, Kazuhiko Yamamoto, Taiichi Saito, and Shinji Matsui. "MAP: Multi-Angled Parallelism for Feature Extraction from Topographical Maps". In: *Pattern Recognition* 24.6 (Jan. 1991), pp. 479–488.

[284] Li Yang, Weihua Huang, and Chew Lim Tan. "Semi-automatic Ground Truth Generation for Chart Image Recognition". In: *Proceedings of the International Workshop on Document Analysis Systems*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 324–335.

[285] Yang Yang, Bin B. Zhu, Rui Guo, Linjun Yang, Shipeng Li, and Nenghai Yu. "A Comprehensive Human Computation Framework: With Application to Image Labeling". In: *Proceedings of the ACM International Conference on Multimedia*. MM '08. Vancouver, BC, Canada: ACM, 2008, pp. 479–488.

[286] David Yarowsky. "Unsupervised Word Sense Disambiguation Rivaling Supervised Methods". In: *Proceedings of the Annual Meeting on Association for Computational Linguistics*. ACL '95. Cambridge, MA, USA: Association for Computational Linguistics, 1995, pp. 189–196.

[287] Hao Ye, Weiyuan Shao, Hong Wang, Jianqi Ma, Li Wang, Yingbin Zheng, and Xiangyang Xue. "Face Recognition via Active Annotation and Learning". In: *Proceedings of the ACM International Conference on Multimedia*. MM '16. Amsterdam, The Netherlands: ACM, Oct. 2016, pp. 1058–1062.

[288] Bowen Yu and Claudio T. Silva. "VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 251–260.

[289] Jan Zahálka and Marcel Worring. "Towards Interactive, Intelligent, and Integrated Multimedia Analytics". In: *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*. VAST '14. Paris, France: IEEE, Oct. 2014, pp. 3–12.

[290] Jan Zahálka, Marcel Worring, and Jarke J. van Wijk. "II-20: Intelligent and pragmatic analytic categorization of image collections". In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2020), pp. 422–431.

[291] Rabah A. Al-Zaidy and Clyde Lee Giles. "A Machine Learning Approach for Semantic Structuring of Scientific Charts in Scholarly Documents". In: *Proceedings of the AAAI Conference on Innovative Applications*. IAAI '17. San Francisco, CA, USA: AAAI Press, 2017, pp. 4644–4649.

[292] Lishi Zhang, Chenghan Fu, and Jia Li. "Collaborative Annotation of Semantic Objects in Images with Multi-Granularity Supervisions". In: *Proceedings of the ACM International Conference on Multimedia*. MM '18. Seoul, Republic of Korea: ACM, 2018, pp. 474–482.

[293] Tianye Zhang, Haozhe Feng, Wei Chen, Zexian Chen, Wenting Zheng, Xiao-Nan Luo, Wenqi Huang, and Anthony K. H. Tung. "ChartNavigator: An Interactive Pattern Identification and Annotation Framework for Charts". In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1.

[294] Yexun Zhang, Wenbin Cai, Wenquan Wang, and Ya Zhang. "Stopping Criterion for Active Learning with Model Stability". In: *ACM Transactions on Intelligent Systems and Technology* 9.2 (Oct. 2017).

[295] Yinan Zhang and Chengxiang Zhai. "Information Retrieval as Card Playing: A Formal Model for Optimizing Interactive Retrieval Interface". In: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. Santiago, Chile: ACM, 2015, pp. 685–694.

[296] Yu Zhang, Bob Coecke, and Min Chen. "On the Cost of Interactions in Interactive Visual Machine Learning". In: *Proceedings of the IEEE VIS Workshop on Evaluation of Interactive Visual Machine Learning Systems*. Vancouver, BC, Canada: IEEE, 2019.

[297] Yu Zhang, Bob Coecke, and Min Chen. "MI3: Machine-Initiated Intelligent Interaction for Interactive Classification and Data Reconstruction". In: *ACM Transactions on Interactive Intelligent Systems* 11.3–4 (Aug. 2021).

[298] Yu Zhang, Martijn Tennekes, Tim de Jong, Lyana Curier, Bob Coecke, and Min Chen. *Using Simulation to Aid the Design and Optimization of Intelligent User Interfaces for Quality Assurance Processes in Machine Learning*. 2021. arXiv: 2104.01129 [cs.HC].

[299] Yu Zhang, Yun Wang, Haidong Zhang, Bin Zhu, Siming Chen, and Dongmei Zhang. "OneLabeler: A Flexible System for Building Data Labeling Tools". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '22. New Orleans, LA, USA: ACM, 2022.

[300] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. "Learning with Local and Global Consistency". In: *Proceedings of the International Conference on Neural Information Processing Systems*. NIPS '03. Whistler, BC, Canada: MIT Press, 2003, pp. 321–328.

[301] Yan Ping Zhou and Chew Lim Tan. "Hough Technique for Bar Charts Detection and Recognition in Document Images". In: *Proceedings of the International Conference on Image Processing*. Vol. 2. ICIP '00. Vancouver, BC, Canada: IEEE, 2000, pp. 605–608.

[302] Zhi-Hua Zhou. "A brief introduction to weakly supervised learning". In: *National Science Review* 5.1 (Aug. 2017), pp. 44–53.

[303] Jingbo Zhu, Huizhen Wang, Eduard Hovy, and Matthew Ma. "Confidence-Based Stopping Criteria for Active Learning for Data Annotation". In: *ACM Transactions on Speech and Language Processing* 6.3 (Apr. 2010).

[304] Xiaojin Zhu. *Semi-Supervised Learning Literature Survey*. Tech. rep. University of Wisconsin–Madison, 2005.

[305] Xiaojin Zhu and Zoubin Ghahramani. *Learning from Labeled and Unlabeled Data with Label Propagation*. Tech. rep. Carnegie Mellon University, 2002.

[306] Xingquan Zhu, Xindong Wu, and Qijun Chen. "Eliminating Class Noise in Large Datasets". In: *Proceedings of the International Conference on Machine Learning*. ICML '03. Washington, DC, USA: AAAI Press, 2003, pp. 920–927.