Advancing Data-Efficient Deep Learning: Non-Parametric Transformers, Active Testing, and In-Context Learning

Jannik Lukas Kossen

New College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2024

Acknowledgements

I want to express my deepest thanks to everyone who has supported me throughout my PhD. Not only have you made this thesis possible, you've made getting here enjoyable.

First and foremost, a heartfelt thank you to my supervisors, Tom Rainforth and Yarin Gal. I have learned so much from both of you over the last years. It has been a real privilege to receive your inspiring supervision, helpful advice, and critical questions. Thank you both for a brilliant time. Tom, thank you for always going the extra mile: for turning 30 minute meetings into three hour deep dives and for late-night pizza-fueled deadline crunch (no pun intended) times. I am deeply impressed by how much you care about every one of your students and the quality of the work that they produce. Yarin, thank you for constantly exposing me to new ideas while also encouraging critical and independent thought. Thank you for teaching me confidence in my abilities and for valuing my opinion in countless projects, collaborations, and discussions.

I would like to express my deepest gratitude to all my collaborators. You have enabled the research that forms the basis of this thesis. I have learned a lot from you and deeply appreciate your contributions to our joint research projects. Firstly, I would like to thank the following current and former members of the OATML research group for their contributions to the research in this thesis and beyond: Sebastian Farquhar, Neil Band, Lorenz Kuhn, Muhammed Razzak, Clare Lyle, Aidan Gomez, Lisa Schut, Shreshth Malik, and Andreas Kirsch. I would also like to thank Danielle Belgrave, Nenad Tomasev, Cătălina Cangea, Eszter Vértes, Andrew Jaegle, and Viorica Patraucean for their support and kindness during my internship at Google DeepMind. Lastly, I want to thank Efi Kokiopoulou, Rodolphe Jenatton, Mark Collier, Basil Mustafa, Xiao Wang, Xiaohua Zhai, Lucas Beyer, Andreas Steiner, and Jesse Berent for their help and goodwill during my internship at Google. I feel incredibly fortunate to have worked with all of you.

I have been very lucky to be part of two labs: Yarin's OATML and Tom's RainML within the wider OxCSML group. To all former and current members: thank you for thoughtful advice, active discussions, and, importantly, fun times. From OATML, in addition to everyone mentioned above, I would like to thank the following current and former members: Joost van Amersfoort, Andrew Jesson,

Milad Alizadeh, Kunal Handa, Lars Holdijk, Kelsey Doerksen, Matthew Kearney, Katrina Dickson, Pascal Notin, Tim Rudner, Lood van Niekerk, Angelos Filos, Panos Tigas, Lewis Smith, Jishnu Mukhoti, Gunshi Gupta, Ruben Weitzman, Luke Melo, Hazel Kim, Sören Mindermann, Jan Brauner, Angus Nicolson, and Yonatan Gideoni. From RainML, I would like to thank Freddie Bickford Smith, Andrew Campbell, Mrinank Sharma, Desi Ivanova, Tim Reichelt, Adam Golinski, and Ning Miao. You have all been a resource of knowledge and joy for me these last years.

Thank you, Freddie, Katrina, and Yonatan for providing helpful comments on a draft of this thesis.

Thank you to Jiatong Han, Gabrielle Berrada, and Aaron Tjandra. It's been a pleasure supervising your MSc projects and I look forward to seeing what your next steps in life and career will be.

I would also like to thank Kristian Kersting, Adam Kosiorek, Karl Stelzner, Claas Voelcker, and Marcel Hussing for encouraging me to apply to Oxford in the first place.

To my friends: thank you for emotional support and joyful distractions. Thank you, Joe(y) and Alli, for being great friends right from the very beginning of the PhD and for letting me "show you the light" (Devlin, 2023) when it comes to machine learning. A thank you to all my other friends who were there for me at some point along the way: Giorgio, Peter, Mo, Freddie, Joost, Andrew, Kunal, and Tom.

I want to thank my family for keeping me in touch with a world beyond machine learning research and for making sure there's always a home to go back to. Thank you, Hanna, Stefan, and Inka for your unconditional support, surprise packages, visits, and phone calls. Thank you, Pero, for providing perspective on what really matters, for the joy you bring, and for football breaks.

Above all, thank you, Jolanda, for being there, with me, through it all. Your love, kindness, encouragements, celebrations, and consolations have made it all possible and worthwhile.

Unfortunately, my memory really is quite terrible, and I am bound to have forgotten to mention someone important in the above acknowledgements. If this is you: I am so sorry, I have no idea how I could forget you, and, importantly, thank you.

Abstract

The creation of ever larger datasets has played an important role in the practical success of deep learning. Unfortunately, in many real-world scenarios, high quality data may be scarce, such that the naive application of deep learning can fall short of expectations. A large variety of prior work aims to remedy this and make deep learning more data-efficient. Such approaches typically rely on one or more of the following high-level strategies: they adjust the model architecture or training to make better use of the existing data, actively control the data creation process to obtain more useful data in the first place, or leverage data from other, indirectly relevant, tasks. In the best case, these methods can drastically improve the performance of deep learning in small data regimes. However, the problem of data-efficiency in deep learning is far from solved and many challenges remain.

This thesis proposes and studies four different approaches to data-efficient deep learning, advancing the state-of-the-art by questioning assumptions made commonly in approaches for data-efficiency. Firstly, we propose Non-Parametric Transformers (NPTs), a data-efficient deep learning architecture that takes the entire dataset as input. This deviates from common deep learning practice and allows NPTs to learn to predict by directly reasoning about interactions between datapoints. NPTs achieve impressive performance, especially on small tabular datasets, where deep learning methods have previously struggled. Secondly, we turn to data-efficiency for model evaluation. While active learning methods reduce the number of labels needed for model training, the cost of labeling for model evaluation is typically ignored without good justification. We address this by introducing two different approaches that allow for label-efficient model evaluation by actively labeling only an informative subset of the datapoints to construct specialized estimates of model performance. Thirdly, we investigate the ability of in-context learning (ICL) in large language models to learn label relationships. There has been significant discussion in the literature about the extent to which ICL actually leverages label information. Our careful study provides novel insights into ICL, revealing both capabilities and limitations.

Contents

1 Introduction

Contents

1.1 Motivation

It has been more than a decade since the burgeoning of deep learning (LeCun et al., 2015; Krizhevsky et al., 2012; Schmidhuber, 2022). Since then, neural networks with steadily increasingly parameter counts have been trained to give better and better performance. While the community disagrees on how far this approach will take us, there is no denying the impressive empirical results achieved by it so far. Today, deep learning architectures are the de facto default choice to address many tasks in major machine learning fields such as computer vision and natural language processing.

Instrumental to the success of deep learning has been the curation of large scale and high quality datasets, such as ImageNet (Deng et al., 2012) for computer vision or BooksCorpus (Zhu et al., 2015) for natural language processing. The increased complexity of deep learning models is only useful if there is sufficient data for them to learn from. If a datasets is too small, deep learning models tend to "overfit": they perfectly memorize the training data but do not predict well on any other inputs (Morgan and Bourlard, 1989; Belkin et al., 2019; Rice et al., 2020). Metaphorically speaking, without the fuel that is data, the engine of deep learning runs dry.

Unfortunately, in many practical scenarios, the number of datapoints available to us may be small. This can be the case, for example, because there is a significant cost associated with the acquisition of each datapoint such as running a real-world experiment or paying an expert to judge each input. Other times, the datasets can be small for no particular reason. The creator of the dataset simply did not collect more data, and we are not in any position to do so either. When the

size of the dataset is small, naive application of default deep learning methods may fall short of expectations.

What can we do in these situations? Sometimes, we may be able to fall back on non-deep machine learning methods, which will often make more efficient use of data. However, for many applications, including popular ones such as image classification or language modeling, the inputs are high dimensional and good predictions require complex non-linear functions to be learned. In these cases, there may be no viable alternative to deep learning. It thus seems a worthwhile goal to try to find strategies that make deep learning more data-efficient and allow its use in situations where data is scarce.

This is the case, in particular, because scarcity is a more common occurrence than the daily life of most deep learning researchers may suggest. They are typically in the privileged position of *choosing* the data that they develop their models on. It is no surprise then that large scale and high quality datasets enjoy widespread popularity across deep learning research communities. However, the machine learning practician is usually in the opposite situation: *given* a particular dataset (or mechanism for generating data) they seek the best machine learning approach. They may often find themselves in a situation where the data that they can feasibly access is not large enough to train deep learning models reliably. It is precisely these situations, where approaches for data-efficiency can enable practitioners to successfully apply deep learning methods.

So how can we improve the data-efficiency of deep learning? Existing approaches typically rely on the following high-level strategies in the pursuit of data-efficiency: they adjust the model architecture or training to make better use of the existing data, they actively control the data creation process to obtain more useful data in the first place, or they leverage data from other, indirectly relevant, tasks. In practice, these strategies may only be successful in symbiosis, and individual approaches may not clearly fall into one category or the other. For example, when actively controlling the data acquisition process, it is desirable to do so in iterative dependence on the model itself – which naturally poses restrictions on which model architecture is

suitable. Further, whether or not an approach is appropriate will often be highly context-dependent. We may not be in a position to control the data acquisition process, we may be restricted in our choice of model architecture, or there may simply not exist any related task data to leverage.

Data-efficiency in deep learning is a large and active area of research. However, while many methods have been proposed, the problem of data-efficiency cannot be considered "solved" and opportunities to improve upon the existing body of work are plentiful.

In this thesis, we propose and study four different approaches to data-efficient deep learning, advancing the state-of-the-art by questioning assumptions made commonly across all three high-level strategies for data-efficiency. Deep learning–based approaches have typically struggled to outperform established baselines on small tabular datasets. In Chapter 3, we propose Non-Parametric Transformers (NPTs), a novel deep learning architecture that defies common deep learning practice by taking the entire dataset as input. NPTs achieve impressive empirical results by learning to predict directly from interactions between datapoints.

While active learning methods reduce the number of labels needed for model training, the cost of labeling for model evaluation is typically ignored without good justification. In Chapters 4 and 5, we introduce two different approaches that reduce the cost of labeling for model evaluation by actively labeling only an informative subset of the datapoints to construct specialized estimates of model performance.

In-context learning (ICL) allows large language models to improve downstream task performance from very few observations without any gradient updates. However, the extent to which ICL actually leverages label information is a topic of active discussion in the literature. In Chapter 6, we thoroughly investigate the extent to which ICL actually *learns* label relationships, revealing both capabilities and limitations.

While all chapters are united in their aim of data-efficiency, each chapter stands on its own in that it discusses a distinct approach for data-efficient deep learning. However, closer inspection reveals, perhaps surprisingly, a similarity between these methods. A standard modeling assumption in deep learning, and much of machine

learning, is that the datapoints are distributed independently and identically when conditioned on the model parameters. A model of the dataset can thus be reduced to a model over independent datapoints. We will later see that, in one way or another, all approaches discussed in this thesis, break this assumption.

NPTs model the dataset in its entirety, allowing them to reason about interactions between datapoints. With active testing, we iteratively adapt the distribution from which we acquire labels for model evaluation, such that datapoints are not distributed identically. And ICL, as a consequence of the language model architecture, makes no assumptions about the independence of datapoints at all – in fact, ICL predictions even depend on the order in which datapoints are observed.

Before we move on to the specifics of this thesis, we would like to offer some general words of caution on the promises of data-efficiency. Methods for data-efficiency are no panacea: their success depends on the task at hand, and, sometimes, one can simply not improve model performance without more data. This is, of course, not to say that the pursuit of data-efficiency is fruitless in most instances. In fact, we even believe that seeking data-efficiency in deep learning can benefit deep learning as a whole – after all, if we can improve model performance with limited data, these benefits may transfer to scenarios where data is abundant, particularly as the tasks attempted by deep learning continue to grow ever more complex. We simply suggest a healthy dose of skepticism about overly strong promises of data-efficiency.

1.2 Outline and Contributions

This section gives an overview of the structure of the thesis and the main contributions made within it. We note that the research that serves as a direct basis for this thesis has previously been published in the proceedings of various machine learning conferences. While I am a "first author" on all of these papers, I am deeply indebted to all my co-authors for their advice and support. Section 1.3 gives a detailed account of their contributions to these papers. In the name of reproducibility, we have made open source code available for all main contributions of this thesis, and we provide links to code repositories at the beginning of each relevant chapter.

The remaining chapters of this thesis are organized as follows: In Chapter 2, we provide an account of useful background material. Chapters 3 to 6 then contain our main contributions. Lastly, Chapter 7 presents closing thoughts and discussion. The following paragraphs give an overview of the contributions made in Chapters 3 to 6 of this thesis.

Chapter 3 Our first main contribution is a data-efficient deep learning architecture that we call Non-Parametric Transformers (NPTs). Unlike established deep learning models, which make predictions in direct dependence on only their parameters and the input's features, NPTs take the entire dataset as input and can learn to predict by directly reasoning about interactions between datapoints. NPTs are an extension of the standard transformer architecture, where multi-head self-attention is applied both within the features of a single input as well as between different datapoints of the dataset, realizing non-parametric prediction with parametric attention. Importantly, NPTs do not *need* to make use of interactions between datapoints for predictions. In part, our motivation is to explore whether, when given the option, gradient-based optimization finds this inductive bias beneficial. Empirically, we find evidence that suggests NPTs can and do learn to make use of direct dependencies between datapoints for prediction. Further, NPTs achieve highly competitive results in the domain of tabular data, where datasets are often small and previous deep learning approaches have failed to match established baselines. This chapter is based directly on the following publication:

> Jannik Kossen*, Neil Band*, Clare Lyle, Aidan N Gomez, Tom Rainforth, and Yarin Gal (2021). "Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Chapter 4 and Chapter 5 Next, we turn to data-efficiency for model evaluation. Model evaluation, more precisely the estimation of the expectation of a model's predictive loss with respect to some data distribution, is an important part of the machine learning pipeline. We need to evaluate model performance, for example, to make sure models are working as intended or to select between different models or

hyperparameters. When labels are expensive to acquire, active learning approaches can reduce the cost of labeling for model *training*. However, the cost of label acquisition for model *evaluation* is typically ignored. With active testing, we address this need and propose methods that reduce labeling costs for model evaluation by actively selectively for which test samples to acquire labels and then forming specialized estimates of model performance. We propose two principled approaches for active testing, one based on an importance sampling Monte Carlo estimator and another that actively learns a surrogate model to form a regression-based estimate. Both methods significantly reduce the cost of model evaluation over a naive baseline that randomly acquires labels for test samples, which is the strategy usually found in practice. These chapters draw directly on the following two publications:

> Jannik Kossen*, Sebastian Farquhar*, Yarin Gal, and Tom Rainforth (2021). "Active Testing: Sample-Efficient Model Evaluation". In: *International Conference on Machine Learning (ICML)*,
>
> Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth (2022). "Active Surrogate Estimators: An Active Learning Approach to Label-Efficient Model Evaluation". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Chapter 6 Lastly, we investigate the extent to which in-context-learning (ICL) in large language models (LLMs) learns label relationships. ICL describes the ability of LLMs to learn novel supervised tasks *in-context*, i.e. by including only a few input-output demonstrations from the task alongside the test query in the input. LLMs are trained on large datasets for next token prediction over natural language, yet, the ability to learn novel supervised language prediction tasks seems to "emerge" from their training at sufficient scale. Notably, ICL works even though the LLM architecture makes no conditional independence assumptions between in-context demonstrations at all. The literature has disagreed quite significantly about the extent to which ICL actually learns novel label relationships, with some works claiming ICL predictions do not even depend on the input labels, while others claim ICL is akin to a general purpose learning algorithm. In this chapter, we present a careful hypothesis-driven investigation of the ability of ICL to learn input-label

relationships, uncovering novel insights as to the capabilities and limitations of ICL. We find that ICL predictions usually do depend on in-context labels, that ICL can learn novel label relationships in context, but that ICL struggles to overcome prediction preferences from pre-training through in-context information, and that it does not consider all information given in-context equally. This chapter is based directly on the following publication:

> Jannik Kossen, Yarin Gal, and Tom Rainforth (2024). "In-Context Learning Learns Label Relationships but Is Not Conventional Learning". In: *International Conference on Learning Representations (ICLR)*.

1.3 Publications and Authorship Statement

The main contributions of this thesis have been previously published in the proceedings of machine learning conferences. While I am a "first author" on all of them, my collaborators have played a significant role that I would like to acknowledge over the following paragraphs.

My supervisors, Yarin Gal and Tom Rainforth, have been instrumental from beginning to end for all projects listed below. Instead of repeating myself regarding their contributions to the individual projects, I here acknowledge that their ideas, feedback, and suggestions for project scoping, experiment design, and writing have been invaluable and have significantly shaped all of my projects.

Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning (Kossen*, Band*, et al., 2021) Neil and I led this project together. We were equally involved in contributing to the codebase and in running experiments. Neil took the main responsibility for implementing the baseline methods for tabular data, and he led the effort to extend NPTs to image classification. I took a lead on designing and implementing our investigation into the capabilities of NPTs, i.e. the semi-synthetic experiments, the row corruption experiments, and the attention visualisations. I wrote most of the initial draft of the main text of the paper. Neil contributed significantly to the writing of the paper, especially but not limited to its lengthy appendix. Clare wrote the

proof for NPTs' equivariance to a permutation of the input datapoints. Aidan gave helpful advice during project meetings.

Active Testing: Sample-Efficient Model Evaluation (Kossen*, Farquhar*, et al., 2021) Seb was involved in the project from the beginning, gave valuable advice throughout many meetings, and gave extensive feedback on the initial draft. I led the project and was responsible for the implementation and execution of all experiments. I wrote the initial draft of the paper but the final version of the paper has benefited significantly from feedback and suggestions from all co-authors.

Active Surrogate Estimators: An Active Learning Approach to Label-Efficient Model Evaluation (Kossen et al., 2022) Seb regularly joined our meetings to give helpful advice. In addition to his abovementioned support as my supervisor, Tom was crucial in developing the XWED acquisition function and analysis of the theoretical properties of the ASE. I led this project and was solely responsible for the implementation and execution of all experiments. I also led the writing of the paper, incorporating helpful suggestions from all co-authors.

In-Context Learning Learns Label Relationships but Is Not Conventional Learning (Kossen et al., 2024) I led this project and was solely responsible for the implementation and execution of all experiments. I also led the writing of the paper, incorporating helpful suggestions from all co-authors.

1.4 Work Not Included in This Thesis

My work over the last years has resulted in three more first-author publications that are worth mentioning at least briefly, as they do fit the theme of data-efficient deep learning. Two of these papers are the outcome of separate industry research internships. Given I can no longer access the codebase or compute necessary to run new experiments or double-check old ones, I have chosen to only mention these publications briefly in this section and not include them as contributions to this

thesis. As for the third paper mentioned below, while I did lead the coding and experimentation, the original method was conceived without me and the writing led by Sebastian, and so I have opted to not include it as a contribution here.

Active Acquisition for Multimodal Temporal Data: A Challenging Decision-Making Task (Abstract) "We introduce a challenging decision-making task that we call active acquisition for multimodal temporal data (A2MT). In many real-world scenarios, input features are not readily available at test time and must instead be acquired at significant cost. With A2MT, we aim to learn agents that actively select which modalities of an input to acquire, trading off acquisition cost and predictive performance. A2MT extends a previous task called active feature acquisition to temporal decision making about high-dimensional inputs. We propose a method based on the Perceiver IO architecture to address A2MT in practice. Our agents are able to solve a novel synthetic scenario requiring practically relevant cross-modal reasoning skills. On two large-scale, real-world datasets, Kinetics-700 and AudioSet, our agents successfully learn cost-reactive acquisition behavior. However, an ablation reveals they are unable to learn adaptive acquisition strategies, emphasizing the difficulty of the task even for state-of-the-art models. Applications of A2MT may be impactful in domains like medicine, robotics, or finance, where modalities differ in acquisition cost and informativeness."

The full text of this paper can be found in:

> Jannik Kossen, Cătălina Cangea, Eszter Vértes, Andrew Jaegle, Viorica Patraucean, Ira Ktena, Nenad Tomasev, and Danielle Belgrave (2023). "Active Acquisition for Multimodal Temporal Data: A Challenging Decision-Making Task". In: *Transactions on Machine Learning Research (TMLR)*.

Three Towers: Flexible Contrastive Learning with Pretrained Image Models (Abstract) "We introduce Three Towers (3T), a flexible method to improve the contrastive learning of vision-language models by incorporating pretrained image classifiers. While contrastive models are usually trained from scratch, LiT (Zhai et al., 2022) has recently shown performance gains from using pretrained

classifier embeddings. However, LiT directly replaces the image tower with the frozen embeddings, excluding any potential benefits from training the image tower contrastively. With 3T, we propose a more flexible strategy that allows the image tower to benefit from both pretrained embeddings and contrastive training. To achieve this, we introduce a third tower that contains the frozen pretrained embeddings, and we encourage alignment between this third tower and the main image-text towers. Empirically, 3T consistently improves over LiT and the CLIP-style from-scratch baseline for retrieval tasks. For classification, 3T reliably improves over the from-scratch baseline, and while it underperforms relative to LiT for JFT-pretrained models, it outperforms LiT for ImageNet-21k and Places365 pretraining."

The full text of this paper can be found in:

> Jannik Kossen*, Mark Collier*, Basil Mustafa, Xiao Wang, Xiaohua Zhai, Lucas Beyer, Andreas Steiner, Jesse Berent, Rodolphe Jenatton, and Efi Kokiopoulou (2023). "Three Towers: Flexible Contrastive Learning with Pretrained Image Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Detecting hallucinations in large language models using semantic entropy

(Abstract) "Large language model (LLM) systems, such as ChatGPT or Gemini, can show impressive reasoning and question-answering capabilities but often 'hallucinate' false outputs and unsubstantiated answers. Answering unreliably or without the necessary information prevents adoption in diverse fields, with problems including fabrication of legal precedents or untrue facts in news articles and even posing a risk to human life in medical domains such as radiology. Encouraging truthfulness through supervision or reinforcement has been only partially successful. Researchers need a general method for detecting hallucinations in LLMs that works even with new and unseen questions to which humans might not know the answer. Here we develop new methods grounded in statistics, proposing entropy-based uncertainty estimators for LLMs to detect a subset of hallucinations – confabulations – which are arbitrary and incorrect generations. Our method addresses the fact that one idea can be expressed in many ways by computing uncertainty at the level of meaning rather than specific sequences of words. Our method works across datasets and

tasks without a priori knowledge of the task, requires no task-specific data and robustly generalizes to new tasks not seen before. By detecting when a prompt is likely to produce a confabulation, our method helps users understand when they must take extra care with LLMs and opens up new possibilities for using LLMs that are otherwise prevented by their unreliability."

The full text of this paper can be found in:

> Sebastian Farquhar*, Jannik Kossen*, Lorenz Kuhn, and Yarin Gal (2024). "Detecting hallucinations in large language models using semantic entropy". In: *Nature*.

2
Background

Contents

In this chapter, we introduce background material relevant to the main contributions of this thesis. Throughout it, we will assume that the reader has a basic level of familiarity with probabilistic machine learning methods, and we refer to popular textbooks such as Murphy (2012), Bishop (2006), and Barber (2012) for further reference. This chapter is split into three sections. In Section 2.1, we first revisit basic machine learning paradigms and their assumptions. A good understanding of this is helpful in the context of this thesis in particular, as some of the approaches that we will discuss blur the lines between paradigms or question fundamental assumptions. In Section 2.2, we then give a brief and modern primer on deep learning. Lastly, Section 2.3 introduces the field of data-efficient deep learning, presenting an overview of fundamental concepts and relevant related work.

2.1 Machine Learning Basics Revisited

This section reviews basic assumptions of machine learning — how we learn, model, and evaluate. While these will likely be already familiar to most readers at some level, we nevertheless feel it is worth reflecting on these fundamentals in this background section, as the approaches discussed in this thesis occasionally touch on core assumptions in machine learning.

2.1.1 Preliminaries

We begin, very practically, by introducing some basic notation and vocabulary. This thesis largely deals with supervised machine learning methods: given a dataset of input-output pairs $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, with inputs $\boldsymbol{x}_i \in \mathcal{X}$ and outputs $y_i \in \mathcal{Y}$, we seek to learn a function $f : \mathcal{X} \to \mathcal{Y}$ from inputs to outputs, $f(\boldsymbol{x}) = \hat{y}$. We will later discuss unsupervised, semi-supervised, or self-supervised methods, where output labels are missing completely, partially, or the there is no clear distinction between labels and inputs.

Typically, the function f will depend on a set of unknown parameters $\boldsymbol{\theta}$: as we change $\boldsymbol{\theta}$, so do our predictions, and we write $f(\cdot; \boldsymbol{\theta})$ to highlight this dependence. In the vast majority of cases, machine "learning" now corresponds to finding the best parameter values $\hat{\boldsymbol{\theta}}$, or a belief distribution over them, from the dataset according to some learning mechanism. We discuss different ways of inferring parameter values from data in Section 2.1.2.

In almost all circumstances, $f(\cdot; \boldsymbol{\theta})$ will only ever be an approximation of the true mechanism which relates inputs to outputs. We therefore refer to $f(\cdot; \boldsymbol{\theta})$ as a model. When the dimension of the parameters $\boldsymbol{\theta}$ is finite, we refer to $f(\cdot; \boldsymbol{\theta})$ as a parametric model. For ease of exposition, we focus on parametric models for now – they are the dominant paradigm in deep learning – and discuss their relation to non-parametric approaches in Section 2.1.3.

When the space of outputs $\mathcal{Y}$ is discrete, i.e. over a set of *classes* $\mathcal{Y} = \{1, \ldots, C\}$, we refer to this as a classification task. Conversely, when the outputs take real values, e.g. $\mathcal{Y} \in \mathbb{R}$, the supervised task is called regression. The space of the inputs $\mathcal{X}$ can be real, discrete, or a mix of both, depending on the setting.

In the *probabilistic* approach to machine learning, which is popular in modern deep learning, instead of learning a mapping to $\mathcal{Y}$ directly, we parameterize a conditional probability distribution over possible output values Y given an input, $p(Y \mid \cdot; \boldsymbol{\theta}) : \mathcal{X} \to \mathcal{P}_{\mathcal{Y}}$. This enables us to learn based on probabilistic principles and express probabilistic uncertainty in our predictions. If desired, we can always

recover a best-guess prediction by determining the output value to which the model assigns maximum probability, $y^* = \arg\max_y p(Y = y \mid \boldsymbol{x}; \boldsymbol{\theta})$.

2.1.2 Principles for Learning

Given a dataset $\mathcal{D}$ and model $f(\cdot; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, there are a variety of strategies for finding parameters which lead to a model that describes the observed data well. Somewhat imprecisely, this process is alternatively referred to as learning, inference, or model training – with the lines sometimes blurred between the learning principle itself and its practical, sometimes approximate, implementation. In this section, we go over three popular principles for learning models from data: empirical risk minimization, frequentist maximum likelihood estimation, and Bayesian inference. Appreciating the commonalities and distinctions between these approaches is important to understand the subtle way in which the approaches of later chapters diverge from standard machine learning practice.

Empirical Risk Minimization A simple and general framework for learning is empirical risk minimization (ERM) (Vapnik and Chervonenkis, 1974; Vapnik, 1991; Shalev-Shwartz and Ben-David, 2014). The empirical risk is the average value of a loss function $\mathcal{L}$ over the training set $\mathcal{D}$,

$$\hat{R}(\boldsymbol{\theta}) = \frac{1}{N} \sum\nolimits_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} \mathcal{L}(f(\boldsymbol{x}_i, \boldsymbol{\theta}), y_i), \tag{2.1}$$

where $N = |\mathcal{D}|$ is the size of the training set and the loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is typically non-negative. ERM now seeks to find those parameters that *minimize* the empirical risk,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta}). \tag{2.2}$$

The empirical risk is colloquially also referred to as the training loss or training error. The most important ingredient in ERM is the choice of loss function, which grades predictions by comparing them to observed outcomes. Popular loss functions include the 0-1-loss for classification, $\mathcal{L}(y, \hat{y}) = \mathbb{1}[y \neq \hat{y}]$, which is 1 for incorrect

predictions and 0 for correct predictions, and the mean-squared error loss for regression, $\mathcal{L}(y, \hat{y}) = (y - \hat{y})^2$, which grows quadratically with the residual error.

In practice, naive ERM objectives are often undesirable as they can lead to overfitting. Overfitting describes when, for a particular set of parameters, the empirical training set risk is low but the average loss on a "test set" of novel observations not included in the training set is high. In other words, a model that overfits is useless because it works really well on the training set – for which we already know the outcomes – but not on novel datapoints – for which we would like to predict the outcomes.

Overfitting is particularly relevant when highly flexible models are trained on limited amounts of data.[1] These are exactly the scenarios of data-efficient deep learning – although we note that even large amounts of data can appear small if the models are flexible enough. In these scenarios, there usually exist values in parameter space, sought out by ERM, which fit the training data with (almost) zero error. However, seeking zero training error typically leads to undesirable overly complex solutions. For example, when the observed outputs are noisy, i.e. the outcomes y are samples from some underlying distribution $p^*(y \mid \boldsymbol{x})$ with finite variance, zero empirical risk directly implies that the learned function f is different from the true function mean, $\mathbb{E}_{Y \sim p^*(\cdot \mid \boldsymbol{x})}[Y]$, which would be the ideal solution under mean squared error loss. Even worse, to interpolate the training samples perfectly, the predictions of f will typically change drastically as we move between two training examples in input space, which leads to bad predictions on any inputs not included exactly in the training set. Intuitively, a solution that is "smoother", i.e. for which the change in outputs is small when the change in inputs is small, is preferable, even if it does not have zero empirical risk.

This motivates minimizing a *regularized* empirical risk, also known as structural risk minimization, which explicitly trades off empirical risk and function complexity

[1] Note that the following "classical" arguments here should be taken with a grain of salt. Especially for deep neural networks, we regularly observe so-called *benign overfitting*: even though the empirical risk is almost zero, the performance on the test set remains very competitive (Belkin et al., 2019; Neyshabur et al., 2019; Bartlett et al., 2020; Hastie et al., 2022).

by introducing a regularizing term $J(\boldsymbol{\theta})$,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta}) + \lambda J(\boldsymbol{\theta}), \tag{2.3}$$

where the scalar hyperparameter λ weights the two terms. A popular regularizer is the squared $L2$-norm of the parameters, $J(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|^2 = \sum_{p=1}^{P} \boldsymbol{\theta}_p^2$, for continuous $\boldsymbol{\theta} \in \mathbb{R}^P$, because high parameter norms usually correspond to more complex functions. Regularized ERM can thus avoid overfitting by penalizing overly complex solutions. It is a general and expressive framework that encompasses many popular learning objectives, including those used to train most deep learning models in general and in this thesis.

Maximum Likelihood Estimation Often, deep learning models predict probability distributions. That is, in the supervised case, they parameterize a conditional probability distribution over output values given an input, $p(Y \mid \boldsymbol{x}; \boldsymbol{\theta})$. A simple approach to training these models is to find the set of parameters $\hat{\boldsymbol{\theta}}$ which best describe the data $\mathcal{D}$. This is known as the principle of maximum likelihood estimation (Fisher, 1912).

The likelihood is the joint probability of the data for a particular set of parameters,

$$p(\mathcal{D}; \boldsymbol{\theta}) = \prod_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} p(Y = y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}), \tag{2.4}$$

which factorizes into a product of individual datapoint probabilities when we assume the datapoints to be independently and identically distributed (more on that later), and where we evaluate the predicted probability distributions at the observed values y_i. Note that, for supervised learning, writing the likelihood as $p(\mathcal{D}; \boldsymbol{\theta})$ is a slight abuse of notation, as Eq. (2.4) is a *conditional* probability of outcomes given inputs and not a joint probability over both inputs and outcomes.

The principle of maximum likelihood now corresponds to maximizing the likelihood function with respect to the parameters $\boldsymbol{\theta}$. For numerical convenience, it is beneficial to optimize the logarithm of the likelihood instead, which does

not affect optima as the logarithm is a monotonically increasing function. The maximum likelihood estimate (MLE) is thus given by

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}; \boldsymbol{\theta}). \tag{2.5}$$

The MLE procedure can be justified from a variety of perspectives. Intuitively, finding the set of parameters under which the data are most probable is a sensible thing to do if we wish to find the best approximation of the true and unknown conditional distribution of outcomes: for example, given two different parameter estimates, the one which has a higher likelihood is, in turn, also more "likely" to generate the observed outcomes in the first place, i.e. it is a better description of the observed data. More theoretically, we can show that the MLE minimizes a divergence between the true outcome distribution and the model, see, e.g. Murphy, 2022, p. 109. Lastly, the MLE enjoys favourable statistical properties and converges to the correct parameter value under the right conditions in the limit of infinite data (Wald, 1949; Cramér, 1999).

We can interpret the MLE as an instance of ERM for probabilistic predictors. Firstly, note that the log likelihood (Eq. (2.5)) decomposes into a sum of log probabilities,

$$\log p(\mathcal{D}; \boldsymbol{\theta}) = \sum\nolimits_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} \log p(Y = y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}), \tag{2.6}$$

which bears strong resemblance to an empirical risk as defined in Eq. (2.1). To identify MLE with ERM, we need write the $\log p(Y = y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta})$ terms as a loss function. This is achieved with the *cross-entropy loss*, $\mathcal{L}(p(Y \mid \boldsymbol{x}_i; \boldsymbol{\theta}), y_i) = -\log p(Y = y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta})$, which is the negative log likelihood the model assigns to the observed outcomes. Maximizing the logarithm of the likelihood is thus identical to minimizing the empirical risk under cross-entropy loss.[2] Similar to our discussion around regularizing ERM objectives, in practice, regularization terms are often added to the maximum log likelihood objective to avoid overfitting.

[2]The missing normalization $1/N$ does not affect optima. Strictly speaking, we also need to redefine the ERM for probabilistic predictions by allowing for loss functions that map from observed outcomes and predicted probability distributions to a loss.

The principle of maximum likelihood is deeply connected to a *frequentist* theory of probability, which defines probability as the long-term frequency with which an observable event occurs (Billingsley, 2013). While this seems natural and straightforward at first, the frequentist approach can become unintuitive, or even problematic, as a tool for statistical inference, see e.g. Berger and Wolpert (1988), Jaynes (2003), and Murphy (2012). For example, as the parameters of models cannot be *observed*, frequentism treats them as fixed quantities and forbids probabilistic reasoning about them. In other words, frequentists categorically cannot speak of the *probability* of their parameter estimates. Instead, frequentists compute so-called confidence intervals, which are probability bounds on the *observed data* relative to the true and unknown parameter value. In general, one should be careful to not misinterpret these as uncertainty statements about the parameter estimates (VanderPlas, 2014), although they practically often serve a similar purpose and can even coincide with Bayesian estimates for some settings.

Bayesian Inference The Bayesian approach to probabilistic inference, while not without flaws, largely avoids such unintuitive surprises. In fact, Bayesian reasoning has been heralded as the extension of logical reasoning to probabilistic quantities (Jaynes, 2003). In a Bayesian framework, a probability is the subjective degree of belief in an abstract statement. This allows for a probabilistic treatment of both observed outcomes and unobserved quantities such as model parameters. The principled uncertainty estimates derived from Bayesian approaches make them attractive to use in situations where data is scarce: for example, we will find Bayesian neural networks (cf. Section 2.3) preferable in our active testing experiments in Chapter 4. We briefly outline the basics of Bayesian inference in the following and refer to textbooks such as those of Jaynes (2003), Robert (2007), Murphy (2012), and Barber (2012), on which the following account is based, for more detail.

Given a dataset of observations $\mathcal{D}$, the Bayesian approach also formulates the likelihood of the dataset, $p(\mathcal{D} \mid \boldsymbol{\theta})$. This likelihood takes the same form as in frequentist inference, except that we now interpret it as the *conditional* probability

of the data *given* a realization of the *random* parameters. In the Bayesian approach, the parameters are random variables about which we are uncertain. This, in turn, allows us to directly reason about our belief in particular parameter values, which is impossible in a frequentist framework.

Bayesian inference requires the specification of a prior distribution on the parameters $p(\boldsymbol{\theta})$. The prior expresses beliefs about the distribution on $\boldsymbol{\theta}$ in the absence of any observations. While this may seem strange at first, there are plenty of practical examples where, either from previous experiments or logical considerations, we can explicitly provide a description of the distribution over $\boldsymbol{\theta}$ we would expect "a priori". And even if we do not have any useful prior knowledge, there are also Bayesian ways of specifying our ignorance. Further, Bayesians may argue that frequentists, instead of avoiding the problem of specifying a prior, end up making an implicit and often suboptimal choice. In practice, the Bayesian prior also serves as a principled way of avoiding overfitting, cf. Section 2.2.2, similar to the regularization terms discussed previously.

Once prior and likelihood are specified, Bayesian inference applies the eponymous Bayes' theorem to infer the *posterior* probability distribution of the parameters given the data, often referred to as just the *posterior*,

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})}. \tag{2.7}$$

The denominator in the above equation is called the *model evidence*, which can be computed from likelihood and prior via marginalization,

$$p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}. \tag{2.8}$$

Instead of a finding a single set of parameters that maximizes the likelihood, the Bayesian approach uses Bayes' theorem to update the prior distribution on the parameters in light of the observed data. Bayesian inference yields a posterior *distribution* over parameter values. This posterior expresses our belief in the value of the parameters – it directly contains our uncertainty over which value is the right

one. Unlike in frequentism, the Bayesian parameter posterior can be used directly to answer any questions regarding our uncertainty in the parameter estimates.

Given we have inferred the parameter posterior, the Bayesian approach methodically applies the machinery of probability theory to derive an expression for making predictions on novel datapoints $\boldsymbol{x}'$,

$$p(y' \mid \boldsymbol{x}', \mathcal{D}) = \int p(y' \mid \boldsymbol{x}', \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) \mathrm{d}\boldsymbol{\theta}, \tag{2.9}$$

which is known as the posterior predictive distribution. When making predictions for novel observations, Bayesians explicitly account for their uncertainty in the parameters via integration. This is different to the frequentists, who typically predict using only the maximum likelihood parameters.

The ease with which the Bayesian approach incorporates and relates uncertainties in observations and parameters is one of its strongest practical selling points. In particular when the amount of data is limited, it can be crucial to account for uncertainty in the value of the parameters. This makes Bayesian approaches particularly popular for data-efficient (deep) machine learning, which we discuss further in the following sections. Conversely, although Bayesians do not need the concept of long-term frequencies to define probabilities, their parameter estimates agree with frequentist maximum likelihood under mild assumptions in the limit of infinite data (Doob, 1949; Van der Vaart, 2000).

For applications, such as linear models, the equations of Bayesian inference can often be computed in closed form. However, for more complex architectures, exact Bayesian inference is typically intractable. A key reason for this is that Bayesian inference requires the computation of integrals in parameter space: both finding the posterior via Bayes' theorem in Eq. (2.7) and consequently computing the posterior predictive in Eq. (2.9) require integration over the parameters. When the parameter space is high-dimensional and the models non-linear, these integrals become very difficult to compute. In practice, Bayesian inference therefore requires approximation schemes such as MCMC (Brooks et al., 2011; Metropolis et al., 1953), Variational Inference (Jordan et al., 1999; Blei et al., 2017), or the Laplace

approximation (MacKay, 1992a).[3] Nevertheless, Bayesian methods can often provide useful uncertainty estimates and improved performance in practice, in particular when the data are scarce.

The IID Assumption Before concluding this section, we would like to highlight a commonality between popular implementations of the learning mechanisms discussed in the above exposition of parametric machine learning: they assume that the data are identically and independently distributed given the parameters, or IID.

From a frequentist perspective, the IID assumption is fundamental and arguably enables the parametric modeling of the data in the first place. Firstly, assuming the data are independent allows us to factorize their joint probability distribution into a product of independent terms. Secondly, assuming the data are identically distributed allows us to re-use the same distribution (with equal parameters $\boldsymbol{\theta}$) for each datapoint. Together, this enables us to learn about $\boldsymbol{\theta}$ from a dataset of observations.

Similarly, in an empirical risk minimization framework, we assume that the empirical risk factorizes over the datapoints and that the same predictor $f(\boldsymbol{x}; \boldsymbol{\theta})$ is applied to each of them. Of course, for ERM with probabilistic predictors and losses, we can directly recover the maximum likelihood objective as discussed before.

Most interesting perhaps is the Bayesian perspective here. While the IID assumption is dogmatic in frequentist modeling, in a Bayesian framework, De Finetti's theorem (De Finetti, 1929) can be used to justify and refine this viewpoint. We refer to Jaynes (2003), Bernardo and Smith (2009), and O'Neill (2009) for modern treatments of this theorem and give an informal and non-rigorous overview next.

We first need to introduce the concept of exchangeability. An infinite sequence of random variables $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N, \ldots$, which we denote with capital letters to distinguish them from observed outcomes, is called *infinitely exchangeable* if, for

[3] In certain situations, we can also learn models that perform Bayesian inference implicitly, cf. our discussion of neural processes in Section 2.2.2.

any N, their joint probability distribution is invariant under permutation π of the order of the random variables,

$$p(\boldsymbol{X}_1 = \boldsymbol{x}_1, \ldots, \boldsymbol{X}_N = \boldsymbol{x}_N) = p(\boldsymbol{X}_1 = \boldsymbol{x}_{\pi(1)}, \ldots, \boldsymbol{X}_N = \boldsymbol{x}_{\pi(N)}). \tag{2.10}$$

Intuitively speaking, if the order in which we observe the data should not matter, the datapoints are exchangeable. De Finetti's theorem states that, for any finite sequence of infinitely exchangeable variables, we can rewrite the joint probability distribution of the data as[4]

$$p(\mathcal{D}) = p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \int \prod_{i=1}^{N} p(\boldsymbol{x}_i \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}, \tag{2.11}$$

where $\boldsymbol{\theta}$ is an appropriately chosen set of parameters with distribution $p(\boldsymbol{\theta})$. Thus, De Finetti's theorem tells us that there is a set of parameters *conditioned* on which the data become IID.[5] Note that, in the Bayesian framework, the joint of the data does not factorize into a product of the marginals – the parameters mediate the dependency between datapoints. In fact, it follows from De Finetti's theorem that existing observations influence our predictions only through an update to our belief about the parameters $\boldsymbol{\theta}$,

$$p(\boldsymbol{x}_{N+1}, \ldots, \boldsymbol{x}_{N+M} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \int \prod_{i=N+1}^{N+M} p(\boldsymbol{x}_i \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \mathrm{d}\boldsymbol{\theta}, \tag{2.12}$$

where we have used the fact that the observations are conditionally independent given the parameter and where the parameter posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ can be inferred via Bayes' theorem. For a single observation $\boldsymbol{x}'$, this simplifies to the posterior predictive introduced in Eq. (2.9).

[4]Technically speaking, we are being imprecise here as the original theorem applies only to Bernoulli random variables. However, there exist many extensions of this theorem to other scenarios, such as continuous multi-dimensional random variables with Normal distribution (Bernardo and Smith, 2009, Ch. 4). Therefore, we follow others such as Murphy (2023) and present the theorem in this imprecise form. Further, while we present the theorem for unconditional probabilities, extensions to conditional observations exist (Bernardo and Smith, 2009, Ch. 4.6.4).

[5]The devil is in the detail here. What exactly $p(\boldsymbol{\theta})$ looks like can be derived under additional assumptions on the random variables. There might not always be a finite-dimensional $\boldsymbol{\theta}$, and, in fact, general forms of De Finetti's theorem tend to be non-parametric (cf. Section 2.1.3), with, for example, latent function values replacing parameters in Eq. (2.12). Also note that $p(\boldsymbol{\theta})$ is *not* the prior, even though it is sometimes used to justify the introduction of a prior distribution. Instead, $p(\boldsymbol{\theta})$ in the theorem is, informally speaking, the empirical distribution of $\boldsymbol{\theta}$ in the limit of infinite data (Bernardo and Smith, 2009).

So in summary, the IID assumption is pervasive across mechanisms for machine learning and can be seen as enabling parametric modeling in the first place. From a Bayesian perspective, De Finetti's theorem can be used to justify that the data are independent and identically distributed when conditioning on the right parameters.

Relevance to This Thesis Most deep learning methods, including those considered in this thesis, are trained using (regularized) maximum likelihood objectives – that is, they are best thought of as frequentist. An exception to this are our investigations into active testing in Chapters 4 and 5, where we find Bayesian deep learning methods advantageous. However, even for the other chapters, Bayesian thinking often provides a useful perspective. For example, there are significant connections between the Non-Parametric Transformer model introduced in Chapter 3 and a class of implicit Bayesian models called Neural Processes (Garnelo et al., 2018b) (see following sections, Chapter 3, and Chapter 7). Similarly, there are claims that in-context learning in large language models implicitly performs Bayesian inference (Xie et al., 2022), which we discuss more in Chapters 6 and 7.

Most relevant perhaps is that the methods studied in this thesis do not assume IID data. Non-Parametric Transformers do not assume that the data are IID and active testing usually does not acquire data in an IID fashion. Lastly, LLMs are not trained for in-context learning directly at all and make no assumptions about the independence or distribution of the input datapoints when used for ICL. While the particular motivations behind breaking IID are different between these approaches, this does suggest a larger theme that leaving IID behind may sometimes be beneficial for achieving data-efficiency.

2.1.3 Parametric and Non-Parametric Models

So far, our discussion has focused on *parametric* models. To some extent, this is justified because parametric models account for the vast majority of deep learning architectures. However, in both Chapters 3 and 6, so-called *non-parametric* models play an important role. We here therefore briefly introduce the non-parametric

approach to machine learning and compare it to its parametric counterpart. The following discussion is based on Rasmussen (2003), Barber (2012), Murphy (2022), Gershman and Blei (2012), and Rainforth (2017), and we refer the interested reader to these sources for further detail.

Parametric Models Revisited First, let us recap the parametric approach to modeling that we have focused on up to this point. In a parametric model f, a finite and fixed number of parameters $\boldsymbol{\theta}$ govern how inputs $\boldsymbol{x}$ are mapped to outputs $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$. A concrete example of a simple parametric model is the following linear regression model: $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \phi(\boldsymbol{x})$, where inputs and outputs are continuous, $\boldsymbol{x} \in \mathbb{R}^D$ and $\hat{y} \in \mathbb{R}$, $\phi : \mathbb{R}^D \to \mathbb{R}^P$ is a (potentially non-linear) feature map, $\boldsymbol{\theta} \in \mathbb{R}^P$ are the parameters, and $\boldsymbol{\theta} \cdot \phi(\boldsymbol{x}) \in \mathbb{R}$ is an inner product. In a parametric model, the training data are used to infer a best-fit value $\hat{\boldsymbol{\theta}}$ of the parameters or a distribution over them. This is the only way in which the training data influence predictions, and after learning about $\hat{\boldsymbol{\theta}}$, we can discard the training data.[6]

Non-Parametric Modeling Non-parametric models work differently. They can be seen – equivalently – as making predictions in direct dependence on the training data, as growing in complexity as more training data become available, or as having no or very few *explicit* parameters but a number of *implicit* parameters growing with the amount of data. We will illustrate these properties for so-called kernel methods (Nadaraya, 1964), a class of models that encompasses many popular non-parametric approaches. Of course, non-parametric modeling has a long history in machine learning and statistics, and there are many important approaches that we do not discuss here for the sake of brevity: other non-parametric methods such as support vector machines (Boser et al., 1992), decision trees (Breiman et al., 1984)[7] such as random forests (Breiman, 2001), Bayesian non-parametric models

[6] Note that this does not imply that it is *only* the parameters which influence predictions. For example, for the linear model, the choice of feature map ϕ obviously has a large influence on the functions that can be learned – and so does the choice of a linear model in the first place.

[7] In practice, decision trees are non-parametric only if the chosen termination criterion does not limit their capacity.

such as Gaussian processes (Rasmussen, 2003) and Dirichlet processes (Ferguson, 1973), or very simple non-parametric approaches such as the k-nearest neighbor classifier (Fix, 1985; Altman, 1992).

From the perspective of this thesis, understanding the behavior and properties of classical non-parametric models is insightful for both Chapter 3 and Chapter 6. In Chapter 3, we will propose and study the Non-Parametric Transformer, a particular deep learning architecture that mixes parametric and non-parametric prediction. And in Chapter 6, we will examine in-context learning in large language models, which can also be seen as a special case of non-parametric prediction.

Any kernel method starts with the definition of a kernel function, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, that measures the similarity between two datapoints. Typically, k is a so-called Mercer kernel, meaning that k is both positive definite and symmetric. A popular example of a kernel is the squared exponential kernel, $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \rho^2 \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2l^2)$, with hyperparameters ρ and l and Euclidean distance $\|\cdot\|$.

The key idea behind kernel methods is that two function values $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ should be similar if the inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are also similar, where input similarity is measured using the kernel. Building on this idea, we can estimate the unknown function value f' for a test input $\boldsymbol{x}'$ given a training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. A simple approach to kernel regression, also called kernel smoothing, is to predict f' as the weighted sum of all observed function values, where the weights are given by (normalized) kernel evaluations,

$$f' = \frac{1}{C} \sum\nolimits_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} k(\boldsymbol{x}_i, \boldsymbol{x}')\, y_i, \quad \text{where} \quad C = \sum\nolimits_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} k(\boldsymbol{x}_i, \boldsymbol{x}'). \tag{2.13}$$

More generally, the *representer theorem* (Kimeldorf and Wahba, 1971; Schölkopf et al., 2001) states that the prediction of many kernel methods, including support vector machines and (the mean prediction of) Gaussian processes, takes the form of a weighted *linear* combination of the kernel evaluated at the training points and the test point,

$$f' = \sum\nolimits_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} \alpha_i\, k(x', x_i), \tag{2.14}$$

where the values of α_i depend on the particular choice of kernel method. Computing the α_i is often much more computationally involved than perhaps suggested by the simple approach of Eq. (2.13) above. For many methods, it can incur significant cost, entailing, for example, the solution of optimization problems or large matrix inversions.

Properties of Non-Parametric Prediction We next discuss a few key properties of non-parametric modeling, which, for kernel methods, can often be read off directly from Eq. (2.14) above.

Firstly, non-parametric models make predictions in direct dependence on the training data. They often require storing the entire training dataset for prediction (the sum in Eq. (2.14) always considers all training points) or their representation grows as the training data grow. The computational costs associated with this can be seen as a practical disadvantage of non-parametric models for large datasets.

Secondly, the complexity of the function represented by non-parametric models grows with the size of the training data. For kernel methods, as the amount of terms increases in Eq. (2.14), so does the complexity of our model. In the absence of exact prior knowledge over a true parametric function underlying the data, the idea that more datapoints justify learning a more complex function is natural and arguably an advantage of the non-parametric approach. Note that this goes beyond the effect of regularization terms and priors that we have discussed previously for parametric models, where the underlying functional form always remains fixed and where, eventually, the complexity of the learned function saturates.

Thirdly, non-parametric methods often have no, or very few, explicit parameters. The explicit parameters that non-parametric models do have are often hyperparameters that are learned in a separate outer loop. For example, for the kernel regression model of Eq. (2.13) the only explicit parameters are the hyperparameters of the kernel, e.g. ρ and l for a squared exponential kernel. The kernel directly determines the *properties* of the modeled function: how rough or smooth it is, how much it fluctuates between datapoints, or if it includes any linear or periodic

trends. Compared to parametric approaches, kernel methods often make it easier to include any such assumptions about the target function.

However, to some extent, they also make it *necessary* to make these choices. This can be problematic in situations where little prior knowledge about the underlying function is available. The choice of kernel and its hyperparameters fully specify the interactions between datapoints, meaning that kernel methods, and non-parametric models more generally, are often surprisingly inflexible in the way that they make predictions. Thus, the correct choice of kernel and its hyperparameters is paramount for achieving good performance with kernel methods. Another consequence of this is that kernel methods can be difficult to use with high-dimensional inputs for which it is hard to manually specify a good similarity function. Conversely, this is something that parametric approaches such as deep neural networks excel at, cf. Section 2.2. Recently, there has been work on combining parametric representation learning with non-parametric prediction (Van Amersfoort et al., 2020; Van Amersfoort et al., 2021; Liu et al., 2020; Wilson et al., 2016), and, in the context of this thesis, both Chapter 3 on Non-Parametric Transformers and Chapter 6 on in-context learning discuss approaches which blur the lines between parametric and non-parametric modeling.

Lastly, we can often also view non-parametric models, and in particular kernel methods, as having a number of implicit parameters growing with the amount of training data. We will only discuss this perspective briefly, and refer to Barber (2012, Ch. 19), Murphy (2022, Ch. 17), or Rasmussen (2003) for further details. In short, a theorem called Mercer's theorem allows us to decompose evaluations of the kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ into inner products of (possibly infinite-dimensional) feature vectors, $\sum_l \phi(\boldsymbol{x}_i)_l \phi(\boldsymbol{x}_j)_l$. Applying this to Eq. (2.14), we can then intuitively view kernel methods as linear models in some feature space where the parameters are constructed implicitly from the training data.

So, in summary, non-parametric models make predictions in direct dependence on the training data, relying on some, often simple, measure of similarity between

training points. They typically have few explicit parameters but the functions represented by them can still grow very complex as the size of the training data grows.

2.1.4 Model Evaluation

Model evaluation is an important part of the machine learning pipeline: we need to evaluate model performance, for example, to assess if a model is working as intended or to choose between competing models or hyperparameters. We here give an overview of different approaches to model evaluation. This is mostly related to Chapters 4 and 5 on active model evaluation, although careful evaluation of model performance is important for Chapters 3 and 6 just as well. Given that Chapters 4 and 5 are fairly brief in their description of conventional approaches to model evaluation – and given that model evaluation truly *is* a basic topic in machine learning – we have chosen to include this discussion here in the background section.

Risk Estimation The simplest – and most popular – approach to model evaluation is to evaluate the model performance on a held-out test dataset. This usually proceeds in the following way: Before doing anything, partition the dataset $\mathcal{D}$ into two non-overlapping subsets, one used for training $\mathcal{D}_{\text{train}}$ and one for testing $\mathcal{D}_{\text{test}}$, $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}} = \mathcal{D}$ and $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$. Use $\mathcal{D}_{\text{train}}$ to *train* the model, e.g. determine the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$ of a deterministic model $f(\cdot; \boldsymbol{\theta})$ via Eq. (2.5).[8] Then, evaluate the average loss of the model predictions on the test set,

$$\hat{R} = \frac{1}{N} \sum\nolimits_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{test}}} \mathcal{L}(f(\boldsymbol{x}_i, \hat{\boldsymbol{\theta}}), y_i), \tag{2.15}$$

where, identical to our discussion on empirical risk, $\mathcal{L}$ is a loss function, with popular choices leading to $\hat{R}$ estimating model accuracy (precisely, 1-accuracy) or cross-entropy loss.[9] In fact, the only difference between Eq. (2.15) and the empirical risk in Eq. (2.1) is that the loss is now computed over the test set instead

[8] For ease of exposition and without restricting generality, we here describe only parametric models.

[9] For simplicity, we here disregard metrics of model performance, such as the F1 score, which do not decompose into a simple sum over the dataset.

of the training set. However, crucially, this means the test set estimate $\hat{R}$ will not overestimate the performance of models that have overfit to the training set.

Intuitively, the larger the size of the test set N[10] the better we can *estimate* the performance of the model. Ideally, we would have an infinitely large test set. We can formalize this by saying that the target quantity we wish to estimate is the model risk

$$r = \mathbb{E}_{\boldsymbol{X},Y}[\mathcal{L}(f(\boldsymbol{X}; \hat{\boldsymbol{\theta}}), Y)], \tag{2.16}$$

which is the expectation of the loss function under the true data-generating distribution.[11]

Monte Carlo theory (Owen, 2013) tells us we can approximate expectations with samples,

$$\mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \int f(\boldsymbol{x})p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{X}_i), \quad \text{with } \boldsymbol{X}_i \sim p. \tag{2.17}$$

Therefore, we can interpret Eq. (2.15) as a Monte Carlo estimate of the risk, $r \approx \hat{R}$.

The statistical properties of Monte Carlo estimates are well-studied and they enjoy favourable asymptotic behavior. Firstly, the estimates are unbiased, which means that the expected value of the *estimator* is the true value, $\mathbb{E}[\hat{R}] = r$, i.e. the average of $\hat{R}$ over infinitely many random test set draws equals the true r. Further, the variance of the estimator, $\mathbb{E}[(\hat{R} - \mathbb{E}[\hat{R}])^2]$ decreases with $1/N$, which tells us how quickly our estimates improve as we increase the number of samples. As the estimator is unbiased, the variance is equal to the overall squared error of the estimator, $\mathbb{E}[(\hat{R} - r)^2]$. The Monte Carlo estimator converges to the true risk in the limit of large N. We refer to Owen (2013) for a detailed introduction to Monte Carlo estimation.

Dividing the data into a fixed train and test set is not without disadvantages. In particular when the dataset is small, there may be significant variance associated with the performance of the model. In such cases, k-fold cross validation is a popular method to get better estimates of model performance (Stone, 1974). First, the data

[10] Note that, for the sake of being consistent with the notation in Chapters 4 and 5 and breaking with notation of previous sections, $N = |\mathcal{D}_{\text{test}}| \neq |\mathcal{D}|$.

[11] We now conform to common statistical practice and take expectations to be with respect to all random (i.e. upper case) variables.

are divided into k non-overlapping folds, $\mathcal{D}_1 \cup \ldots \mathcal{D}_k = \mathcal{D}$. Then, the following is repeated k times: In iteration i, train the model on $\mathcal{D}_1 \cup \mathcal{D}_{i-1} \cup \mathcal{D}_{i+1} \cup \ldots \mathcal{D}_k$, i.e. all folds except $\mathcal{D}_i$, and then evaluate the model on the test fold $\mathcal{D}_i$. The average of the test set errors across iterations is used as an estimate of model performance, reducing dependence on a particular fold. However, for many deep learning scenarios, cross-validation is not feasible as the cost of training multiple models is too large.

Monte Carlo estimators, which rely on arguments about other possible realizations of the data, are fundamentally frequentist, as Bayesians take the observed data as fixed and given. Therefore, risk estimation is a frequentist method for model evaluation. Next, we will discuss Bayesian approaches to model evaluation.

Bayesian Model Comparison As we have previously discussed, a common motivation for model evaluation is to choose between a selection of models, in other words, attempting to find the one that performs best. In a Bayesian framework, we can compare two models, $\mathcal{M}_1$ and $\mathcal{M}_2$, by computing the Bayes factor

$$\frac{p(\mathcal{M}_1 \mid \mathcal{D})}{p(\mathcal{M}_2 \mid \mathcal{D})} = \frac{p(\mathcal{D} \mid \mathcal{M}_1)p(\mathcal{M}_1)}{p(\mathcal{D} \mid \mathcal{M}_2)p(\mathcal{M}_2)}, \tag{2.18}$$

where $p(\mathcal{D} \mid \mathcal{M}_i)$ is the marginal likelihood, or evidence, given by Eq. (2.8), and $p(\mathcal{M}_1)$ and $p(\mathcal{M}_2)$ are priors over the model choice. If we have no a priori preference for one model over another, we can set $p(\mathcal{M}_1) = p(\mathcal{M}_2)$, such that the terms cancel. If the ratio in Eq. (2.18) is larger than 1, we prefer model $\mathcal{M}_1$ over $\mathcal{M}_2$ and vice versa if the ratio is smaller than 1.

While this approach often works well for choosing between hyperparameters within a single model class, results are less clear cut when comparing different models entirely. Concretely, Bayesian model comparison can be very sensitive to the parameter prior $p(\boldsymbol{\theta})$ (hiding in the evidence, cf. Eq. (2.8)) (Domingos, 1999; Gelman, 2011). Similarly, Lotfi et al. (2022) highlight that the marginal likelihood tells us how likely it is the prior generates the observed data but not how good the posterior predictive predicts on novel, withheld observations.[12]

[12]Interestingly, this is not where the discussion ends. For one, there are close connections between leave-one-out-cross validation, i.e. $k = N$, and the Bayesian marginal likelihood (Fong

So in summary, the frequentist approach of evaluating model performance on a held-out validation set is both robust, provided the held-out set is large enough, and popular in machine learning research. It can be used to evaluate and compare any type of predictive model, both Bayesian and non-Bayesian. And in machine learning research practice, even Bayesians often rely on held-out sets to compare model performance to avoid the subjective nature of the Bayesian approach.

2.2 Deep Learning

This section starts with a brief introduction to deep learning fundamentals. Then, we discuss various popular deep learning architectures that are relevant to this thesis. We refer the reader to Goodfellow et al. (2016) and LeCun et al. (2015) for additional material on the topic.

2.2.1 Fundamentals

The "deep" in deep learning refers to the repeated application of similar units of computation. Nowadays, the most popular deep learning architecture is the deep neural network (DNN). A suitably deep and wide (see below) DNN is highly flexible and can, in practice and in theory (Leshno et al., 1993), approximate a very large class of functions.

A popular task for DNNs is classification, and we will use this as our example scenario throughout this section. For classification, the DNN typically parameterizes the distribution over class labels $p(Y = y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta})$, that is, the DNN is a parametric and probabilistic model. It is usually trained using a maximum likelihood objective, or, equivalently, empirical risk minimization under cross-entropy loss. To combat overfitting, the objective is often augmented with regularization terms such as L2 penalties, weight decay (Hanson and Pratt, 1988; Loshchilov and Hutter, 2019), or dropout (Srivastava et al., 2014; Gal and Ghahramani, 2016).

and Holmes, 2020). But similar to the arguments of Lotfi et al. (2022), frequentist cross-validation has been criticized as not estimating the desired quantity (Bates et al., 2023).

The Multilayer Perceptron The simplest example of a deep learning architecture is the multilayer perceptron (MLP) (Rosenblatt, 1958). The definition for what exactly an MLP is and how its parameters are trained has evolved significantly over the years (Linnainmaa, 1970; Ivakhnenko, 1971; Amari, 1967; Schmidhuber, 2022). Here, we present a modern perspective on MLPs. In their current-day form, MLPs are still a fundamental building block of almost all deep learning architectures.

The series of computations defining the MLP is fairly straightforward. The MLP repeats the following two operations across each of its layers: a learned linear map followed by a fixed element-wise non-linearity. More concretely, for the example of a classification task with inputs $\boldsymbol{x} \in \mathbb{R}^d$ and target labels $y \in \{1, \dots, C\}$, an MLP with K layers computes

$$\boldsymbol{z}_1 = \sigma(\boldsymbol{W}_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1), \quad \text{with } \boldsymbol{W}_1 \in \mathbb{R}^{d \times d_1} \text{ and } \boldsymbol{b}_1 \in \mathbb{R}^{d_1}, \tag{2.19}$$

$$\boldsymbol{z}_2 = \sigma(\boldsymbol{W}_2 \cdot \boldsymbol{z}_1 + \boldsymbol{b}_2), \quad \text{with } \boldsymbol{W}_2 \in \mathbb{R}^{d_1 \times d_2} \text{ and } \boldsymbol{b}_2 \in \mathbb{R}^{d_2}, \tag{2.20}$$

$$\dots$$

$$\boldsymbol{z}_{K-1} = \sigma(\boldsymbol{W}_{K-1} \cdot \boldsymbol{z}_{K-2} + \boldsymbol{b}_{K-1}), \quad \text{with } \boldsymbol{W}_1 \in \mathbb{R}^{d_{K-2} \times C} \text{ and } \boldsymbol{b}_{K-1} \in \mathbb{R}^{C}, \tag{2.21}$$

$$\boldsymbol{z}_K = \text{softmax}(\boldsymbol{z}_{K-1}). \tag{2.22}$$

Here, $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$, with $i \in \{1, \dots, K-1\}$, are the weights and biases of the MLP, $\boldsymbol{W} \cdot \boldsymbol{z}$ denotes matrix multiplication, and $\sigma(\cdot)$ is an element-wise non-linear *activation function*. In the common case, the activation function is fixed and has no learnable parameters. Then, the parameters $\boldsymbol{\theta}$ of the MLP are the collection of all its weights and biases, $\boldsymbol{\theta} = \{(\boldsymbol{W}_i, \boldsymbol{b}_i)\}_{i=1}^{K-1}$. In modern deep learning architectures, the total number of learnable parameters varies from millions (Krizhevsky et al., 2012) to hundreds of billions (Anil et al., 2023).

In practice, the dimensionality of the intermediate *hidden states*, $\{\boldsymbol{z}_2, \dots, \boldsymbol{z}_{K-1}\}$, also known as *activations* or *neurons*, is often similar, e.g. $\boldsymbol{W}_k \in \mathbb{R}^{h \times h}$ are square matrices of fixed shape across most layers. The *hidden dimension*, h, is then also referred to as the *width* of the MLP, while K corresponds to its *depth*.

The softmax function ensures that the output of the MLP, $\boldsymbol{z}_k \in \mathbb{R}^C$, can be interpreted as a categorical probability distribution over classes. For a vector-valued

input $\boldsymbol{z} \in \mathbb{R}^C$, it is defined as $\text{softmax}(\boldsymbol{z}) = \exp(\boldsymbol{z})/(\sum_{d=1}^{C} \exp(\boldsymbol{z}_d)) \in \mathbb{R}^C$. After application of the softmax function, we have that $0 \leq z_d \leq 1$ for $d \in \{1, \ldots, C\}$ and $\sum_{d=1}^{C} \boldsymbol{z}_d = 1$. Therefore, we can interpret $\boldsymbol{z}_k$ as a vector of probabilities that models the conditional distribution of output classes given inputs, $\boldsymbol{z}_k = p(\boldsymbol{y} \mid \boldsymbol{x}; \theta)$.

Activation Functions The non-linear activation function, σ, is a crucial component of the MLP. Without it, the overall function learned by the MLP would collapse to a linear function, as the repeated application of linear functions gives a linear function. Historically, activation functions such as the hyperbolic tangent or the sigmoid function, $1/(1 + \exp(-x))$, have been used. These were then mostly replaced by the rectified linear unit (ReLU) (Fukushima, 1975; Nair and Hinton, 2010), which is defined as $\sigma(x) = \max(0, x)$, suppressing negative inputs. Unlike the hyperbolic tangent or sigmoid function, the ReLU does not "saturate" for large inputs, and thus avoids the "vanishing gradient" problem which hinders learning. However, the ReLU activation is not without flaws: its full suppression of negative inputs can lead to an undesirable phenomenon known as *dying neurons* (Trottier et al., 2017; Agarap, 2018; Lu et al., 2019). Currently, popular activation functions often take the form of smooth approximations to the ReLU, sometimes additionally introducing learnable parameters or multiplicative interactions (Hendrycks and Gimpel, 2016; Ramachandran et al., 2017; Dauphin et al., 2017; Shazeer, 2020).

Training Neural Networks DNNs, including MLPs and more modern architectures (see next section), are typically trained with the following recipe. First, the weights and biases of the network are initialized randomly by drawing from some fixed distribution; for the weights, this is often a uniform distribution that depends on the input and output dimensions of the layer (Glorot and Bengio, 2010; He et al., 2015). The parameters of the neural network are then optimized numerically using *stochastic mini-batch gradient-descent*. Firstly, "stochastic mini-batch" refers to the fact that we are not directly minimizing the loss over all training datapoints at once. Instead, we divide the dataset into randomly chosen, equally-sized, non-overlapping subsets called *mini-batches*, $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_M$, with $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ for $i \neq j$.

For each mini-batch m, we update the parameters by taking a *gradient descent* step on that mini-batch. This means, we compute the gradient of the empirical risk on the mini-batch (which is, confusingly, also often called the loss) with respect to the current parameters,

$$\nabla_{\boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta}, \mathcal{D}_m) = \nabla_{\boldsymbol{\theta}} \left(\frac{1}{|\mathcal{D}_m|} \sum\nolimits_{(\boldsymbol{x}_m, y_m) \in \mathcal{D}_m} \mathcal{L}(f(\boldsymbol{x}_m; \boldsymbol{\theta}), y_m) \right). \tag{2.23}$$

This gradient tells us the direction in high-dimensional parameter space in which the mini-batch empirical risk of the model increases the most. We then adjust the model parameters by taking a step in parameter space in the opposite direction, $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta}, \mathcal{D}_m)$, obtaining a new set of parameters for which the loss should be smaller. Here, ϵ is called the *learning rate*.

Once we have repeated this procedure for all mini-batches in the dataset, iteratively updating our parameters to decrease mini-batch empirical risks, we have completed an *epoch* of training. One typically trains for multiple epochs, creating new random mini-batches for each one.

As DNNs are very flexible models, the empirical risk on the training set can be a misleading measure for training progress due to overfitting. A common way to combat this is with a held-out validation set, which is only used to obtain an unbiased estimate of the model's performance. Training is stopped when model performance on this validation set no longer improves.

Learning Rates The scale of the learning rate ϵ, i.e. the step size of our parameter updates, is a hyperparameter that must be chosen carefully. If ϵ is too small, the loss of our model does not change meaningfully. However, if we take ϵ steps that are too large, the optimization might become unstable as the local gradient estimate is no longer representative of how the model risk actually changes after the update. It is further common to change the learning rate during training, e.g. linearly increase it during the first update steps (e.g. Vaswani et al. (2017) and Goyal et al. (2017)) and then decrease it again over the training run (e.g. Loshchilov and Hutter (2017) and DeVries and Taylor (2017)). In practice, learning rates need to be tuned depending on the dataset and model architecture.

Optimizers In modern deep learning, the parameter update we are taking is typically not the naive gradient update $\epsilon \nabla_{\boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta}, \mathcal{D}_m)$. Instead, we rely on more carefully crafted update rules specified by *optimizers* such as Adam (Kingma and Ba, 2015), AdamW (Loshchilov and Hutter, 2019), or, less popularly these days, Adadelta (Zeiler, 2012), Adagrad (Duchi et al., 2011), or RMSProp (Hinton et al., 2012). Optimizers seek to improve model training over naive gradient descent by making learning rates adaptive to individual parameters, by approximating higher order gradients, or by stabilizing gradients with running estimates of gradient momentum or variance. In comparison to naive gradient descent, optimizers can improve model training, even though they require additional hyperparameter tuning on top of the base learning rate.

Backpropagation Important to the practical success of DNNs was the development of the backpropagation algorithm, which is a special case of an algorithm known as reverse-mode automatic differentiation (Linnainmaa, 1976; Rumelhart et al., 1985; Baydin et al., 2018; Griewank, 2012). Backpropagation allows us to efficiently compute exact numerical values for the gradients of our risk estimates with respect to the parameters of the model. These gradients are computed automatically through clever application of the chain rule of differentiation to the layers of the neural network. The user simply specifies the computation that is parameterized by the model, e.g. the MLP of Eqs. (2.19) to (2.22), and then, gradients are automatically computed with the so-called *backward pass* through the model. The emergence of simple-to-use modern deep learning programming frameworks that automate backpropagation, such as PyTorch (Paszke et al., 2019) or TensorFlow (Martín Abadi et al., 2015), has made deep learning accessible and contributed significantly to its popularity.

2.2.2 Modern Deep Learning Methods

The empirical success of deep learning is in no small part due to research into novel model architectures beyond MLPs such as Convolutional Neural Networks (CNNs) or

Transformers (see below). While MLPs usually still play an important role in these models, they are combined with other operations, such as convolutions or attention, that are crafted to exploit invariances or structure in the task data. In this section, we will introduce modern approaches to deep learning that are relevant to this thesis, including CNNs, Transformers, Bayesian Neural Networks, and Neural Processes.

Convolutional Neural Networks (CNNs) After early work in the 1990s (e.g. LeCun et al. (1998)), CNNs rose to popularity in the early 2010s when the AlexNet CNN (Krizhevsky et al., 2012) won the ImageNet image classification competition (Deng et al., 2012) by a large margin against other non-CNN approaches. CNNs soon became the default architecture for vision tasks and a large variety of improved CNN architectures were proposed, e.g. by Szegedy et al. (2015), He et al. (2016), and Howard et al. (2017). Recently, Vision Transformer models have outperformed CNNs in scenarios where very large datasets are available (Dosovitskiy et al., 2021). However, image classification with CNNs remains one of the most popular default deep learning scenarios, and it is the scenario assumed by many papers on data-efficient deep learning as well. Chapters 4 and 5 on active testing also focus chiefly on image classification with CNNs.

We next give a very brief sketch of the CNN architecture and refer to the above works or Murphy (2022, Ch. 14) for technical details. The eponymous operation of the CNN is the *convolution*, which is a learned linear map from a patch of the image input to a vector that is applied in sliding-window fashion to all patches of the input image.[13] This exploits the idea that information in images is, to some extent, translation-invariant (e.g. the target label does not change when the object moves to a different location in the image). Similar to the MLP architecture, convolutions are stacked across multiple layers and interspersed with non-linear activation functions. While earlier layers often learn simple functions, such as edge detectors, more complex representations emerge in the deeper layers of the

[13]Technically speaking, CNNs are usually implemented using a cross-correlation and not a convolution (Cohen et al., 2018). This does not affect the functionality of CNNs with learned kernels because a convolution is identical to a cross-correlation with the kernel transposed.

CNN, see e.g. Olah et al. (2018). CNNs additionally usually contain so-called *pooling layers* which are also applied in a sliding-window fashion but used to reduce dimensionality of the input, e.g. max pooling retains only the maximum activation in a local neighborhood of features (Yamaguchi et al., 1990). The last layers of a CNN are typically MLPs, which gradually reduce the dimensionality of the hidden activations to the size of the output label space.

Transformers Since their introduction by Vaswani et al. (2017), the Transformer architecture has become the dominant paradigm for natural language processing (NLP). Prior to the Transformer, recurrent neural networks (RNNs), such as the LSTM (Hochreiter and Schmidhuber, 1997), were dominant. Unlike RNNs, which aggregate previous observations into one or multiple hidden states, Transformers rely on blocks of multi-head self-attention operations, which explicitly *attend* to all previous observations in the sequence. We formally introduce the building blocks of the Transformer architecture in Chapter 3, together with our own Non-Parametric Transformer (NPT) model.

The initial Transformer architecture used an *encoder-decoder* setup, which is useful for tasks such as translation where there is an obvious input-output relation between two sequences to be learned. A number of variants of the Transformer architecture have been proposed, succeeding the encoder-decoder configuration in popularity. Devlin et al. (2019) introduced the BERT model, which relies only on an encoder, learning to predict the value of tokens (words or word fragments) that were masked out in the input. Currently, so-called *decoder-only models* (Liu et al., 2018; Radford et al., 2018) are the most popular. These fully autoregressive models – trained to predict a distribution over the next token given all previous observed or predicted tokens – are the base architecture for the current generation of large language models (Radford et al., 2019; Chowdhery et al., 2022; Hoffmann et al., 2022; Zhang et al., 2022a; Touvron et al., 2023a). Decoder-only models are the topic of our investigation in Chapter 6.

While Transformers were originally introduced for NLP, they have found fruitful application in other domains. As previously mentioned, Dosovitskiy et al. (2021) demonstrate impressive image classification performance with their Vision Transformer model. Jaegle et al. (2021) and Jaegle et al. (2022) propose general purpose transformer architectures that work well across a variety of input domains. In Chapter 3, we introduce the NPT architecture, a generalization of the Transformer architecture targeting tabular data. NPT-like models have since also been shown to work well with biological data such as protein sequences (Rao et al., 2021; Notin et al., 2023).

Bayesian Deep Learning Bayesian Deep Learning (BDL) describes a family of models that apply the machinery of Bayesian reasoning to deep neural networks as described in Section 2.1.2. First, a prior distribution over the parameters of the Bayesian neural network (BNN), $p(\boldsymbol{\theta})$, is typically specified.[14] Given a dataset $\mathcal{D}$, the prior is updated to the posterior via Bayes' rule, Eq. (2.7), and predictions are then made with the posterior predictive, Eq. (2.9) – both steps generally have to be approximated for complex Bayesian models such as BNNs.

Advocates of BNNs often promote their automatic model complexity (and hyperparameter) selection abilities, which ideally help avoid overfitting and thus improve predictive performance (MacKay, 1992b). Further, BNN predictions may often become *uncertain* when asked to predict on a point that is different from previous training data. This uncertainty is related to *epistemic* uncertainty (Der Kiureghian and Ditlevsen, 2009; Kendall and Gal, 2017), which represents uncertainty in predictions due to uncertainties in the parameter posterior. Epistemic uncertainty is often presented as a crucial advantage of BNNs over their deterministic counterparts, for example, in applications where data-efficiency is important. This is the key reason BDL is important to this thesis, and we will discuss epistemic uncertainty further in Section 2.3.

[14] How to meaningfully specify a prior for BNNs is a topic of active debate, see e.g. Wenzel et al. (2020), Izmailov et al. (2021), Fortuin et al. (2022), and Tran et al. (2022).

Unfortunately, exact inference is intractable for BNNs, as the high-dimensional integrals in parameter space required by the Bayesian approach cannot be computed exactly. Research in BDL therefore often concentrates on developing methods for *approximative* Bayesian inference, such as variational inference (Hinton and Van Camp, 1993; Graves, 2011; Blundell et al., 2015b), the Laplace approximation (MacKay, 1992a; Daxberger et al., 2021), or Monte Carlo sampling methods (Neal, 1995; Cobb and Jalaian, 2021; Izmailov et al., 2021). Recently, Gal and Ghahramani (2016), Maddox et al. (2019), and Lakshminarayanan et al. (2017) and others have proposed BNNs which can be trained with simple modifications to standard deterministic model training. Important to us here are radial BNNs (Farquhar et al., 2020), a particular type of variational inference–based BNN, which we rely on in our active testing experiments in Chapter 4.

We further use deep ensembles (Lakshminarayanan et al., 2017), which are unweighted ensembles of independently trained neural networks that differ only in the random seed used to initialize model weights and shuffle training data. While deep ensembles are not obviously Bayesian, there is some discussion in the literature as to whether aspects of them can be practically interpreted as such (Wilson and Izmailov, 2020; Wild et al., 2024). Importantly to us, deep ensembles allow for the computation of quantities that can be related to Bayesian uncertainties, are easy to implement, and obtain good performance in practical settings.

Even with approximate inference, BNNs typically increase computational costs at training and/or inference time. Unlike deterministic NNs, BNNs need to model a distribution over weights, which increases parameter counts and/or makes sampling approximations necessary. This makes BNNs difficult to scale to large models and unattractive to practitioners. Therefore, they have remained a somewhat niche topic in deep learning, with research efforts focusing mostly on applications to data-efficiency, such as active learning (Houlsby et al., 2011; Gal et al., 2017), where good uncertainties should be most helpful.

Neural Processes An exciting development in Bayesian deep learning is a family of models called Neural Processes (NPs) (Garnelo et al., 2018b; Garnelo et al., 2018a; Kim et al., 2019). NPs can be seen as a deep learning version of the classic Gaussian process (GP) model (Rasmussen, 2003). In a GP a kernel function, see Section 2.1.3, directly specifies how the training data influence predictions on test data, with kernel hyperparameters affording only limited flexibility. NPs on the other hand *learn* a map from a set of training points to a test point prediction using a highly flexible and carefully constructed DNN architecture. That is, each input to the NP is a dataset (not a datapoint) and NPs are trained over a collection of datasets. When these datasets are drawn from the prior of some Bayesian model such as a Gaussian process, NPs have been shown to recover the predictive posterior distribution of the underlying Gaussian process with low error (Müller et al., 2022). Note that, while the parameters of the NP can be fully deterministic, NP predictions can be seen as a form of implicit, or amortized, Bayesian inference in some circumstances (Gordon et al., 2019; Nguyen and Grover, 2022).

In the context of this thesis, NPs have important links to both Chapter 3 and Chapter 6. Similar to the NPT architecture that we introduce in Chapter 3, NPs take entire datasets as input to the model. However, NPTs can be trained on single datasets. Like NPs, in-context learning in large language models, which we discuss in detail in Chapter 6, also learns a direct map from a training set of datapoints to a test query across a variety of task. And similarly to NPs, in-context learning has been compared to implicit Bayesian inference (Xie et al., 2022), which we will discuss in more detail in Chapter 7.

2.3 Data-Efficient Deep Learning

In this section, we will give a brief introduction to methods for data-efficient deep learning. We will cover so-called active learning methods, inductive biases and priors, semi-supervised learning, finetuning and transfer learning, and in-context learning. While the literature often discusses these approaches in isolation, we here

present them together, highlighting the fact that they are united in their aim of data-efficiency, and that, sometimes, much can be gained from combining them.

2.3.1 Active Learning

Supervised machine learning requires a training set of input output pairs, $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. In many practical applications, the cost of acquiring the labels y is much higher than the cost of obtaining inputs $\boldsymbol{x}$. This is the case for example, when determining y requires expert knowledge (e.g. we need to pay a medical expert to determine an illness) or expensive experiments (e.g. a compute-intense simulation or an actual experiment in a laboratory). Active learning aims to reduce the number of labels needed for model training by constructing an *interactive loop* in which *the model* determines for which input to acquire labels next (Atlas et al., 1990; Seung et al., 1992; MacKay, 1992d; Lewis, 1995). That is, we do not acquire labels for random inputs, but instead use the model to judge which inputs are informative given its current state of knowledge. We then update the model given the latest $(\boldsymbol{x}, y)$ observations and repeat. Related to this, in Chapters 4 and 5 we develop methods for *evaluating* models when labels are expensive.

We only give a brief overview of active learning methods here and refer to Settles (2010) for a classic overview of active learning methods and to Ren et al. (2021) for an overview of deep active learning methods. In particular, we focus on *pool-based Bayesian active learning*. *Pool-based* means that we assume the availability of a large and fixed set of inputs (but not labels), opposed to a *streaming* setting, where the inputs themselves are observed sequentially. We also focus on *Bayesian* active learning, as the application of Bayesian reasoning leads to a principled framework for data acquisition.

Bayesian Active Learning Bayesian active learning (MacKay, 1992d; MacKay, 1992c; Lawrence et al., 2002; Houlsby et al., 2011) typically uses the criterion of *expected information gain* to acquire labels. For example, given a model with

parameter prior $p(\boldsymbol{\Theta})$, the expected information gain (EIG) in $\boldsymbol{\Theta}$ is defined as

$$\mathrm{EIG}_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \mathbb{E}_{Y \sim p(\cdot|\boldsymbol{x})}[\mathrm{IG}_{\boldsymbol{\Theta}}(\boldsymbol{x}, Y)], \tag{2.24}$$

with

$$\mathrm{IG}_{\boldsymbol{\Theta}}(\boldsymbol{x}, y) = \mathbb{H}[p(\boldsymbol{\Theta})] - \mathbb{H}[p(\boldsymbol{\Theta} \mid y, \boldsymbol{x})]. \tag{2.25}$$

Here, $\mathbb{H}[p(\boldsymbol{\Theta})] = -\mathbb{E}_{\boldsymbol{\Theta} \sim p(\cdot)}[\log p(\boldsymbol{\Theta})]$ is the Shannon entropy (Shannon, 1948; MacKay, 2003), which can be understood as the uncertainty in the random parameters. The parameter information gain (IG) describes how much our uncertainty in the parameters decreases after observing label y for input $\boldsymbol{x}$. As we do not know y, we evaluate the parameter EIG instead, which substitutes a particular value for y with our expectation of it under our model. The parameter EIG describes how much we would *expect* our uncertainty in the parameters to decrease when acquiring the unknown label of input $\boldsymbol{x}$. Bayesian active learning acquires the particular input $\boldsymbol{x}^*$ for which the EIG is maximal. We then update our parameter beliefs to account for the additional observation. For the next iteration, we again evaluate the EIG at all remaining unlabeled points in the pool, replacing the parameter prior with the posterior from the previous round.

Equation (2.24) can be rewritten an expected reduction in *predictive* uncertainty, $H[Y|\boldsymbol{x}] - \mathbb{E}_{\boldsymbol{\Theta} \sim p(\cdot)}[H[Y \mid \boldsymbol{x}, \boldsymbol{\Theta}]]$, which can be easier to calculate in practice. In machine learning, this version of the parameter EIG is often called the BALD (Bayesian Active Learning by Disagreement) equation (Houlsby et al., 2011). In Chapter 5, we rely on a variation of the BALD equation to actively learn a surrogate model for active model evaluation.

Active learning can be seen as a special case of Bayesian optimal experimental design (Lindley, 1956; Chaloner and Verdinelli, 1995; Sebastiani and Wynn, 2000; Rainforth et al., 2024) which formalizes maximizing the expected information gained from an experiment.

An early example of *deep* Bayesian active learning is the work of Gal et al. (2017); they estimate the parameter EIG of Monte Carlo Dropout BNNs (Gal and

Ghahramani, 2016) for active learning of image classification tasks. We again refer to Ren et al. (2021) for a survey of the related work in deep Bayesian active learning.

Bias in Active Learning Of particular importance to this thesis is the work of Farquhar et al. (2021), who study bias in empirical risk estimators when samples are actively acquired. A consequence of actively acquiring datapoints with an acquisition function is that the training objective, typically an empirical risk such as Eq. (2.2), is no longer an unbiased estimator of the true risk. A priori, one might expect this bias to be detrimental to the performance of the actively learned model. Farquhar et al. (2021) investigate this and introduce an estimator that removes the bias introduced by (stochastic) active acquisition. They find that, while for linear models the unbiased estimator performs better, deep learning models often actually *benefit* from the bias introduced by active acquisition because it acts as an implicit regularizer that encourages generalization. However, in Chapter 4, we demonstrate that this estimator is very useful for the active evaluation of model performance.

Prediction-Oriented Active Learning Recently, Bickford Smith et al. (2023) have proposed a paradigm shift for Bayesian active learning, building on early work by MacKay (1992c) and MacKay (1992d). They illustrate that the default EIG objective, which maximizes information gained in *parameters* can sometimes lead to negligible improvements in *predictions*. Bickford Smith et al. (2023) thus introduce EPIG (Expected Predictive Information Gain), which aims to directly reduce the model's expected *predictive* uncertainty over a random test time input drawn from the training set or a particular test set. They argue that EPIG is more robust than parameter EIG and better aligned with the fundamental goals of active learning.

Interestingly, the EPIG acquisition objective can be evaluated using only posterior predictives, and it does not require knowledge of an explicit parameter posterior. In the context of this thesis, EPIG is therefore an interesting topic of discussion for in-context learning in models such as neural processes (related to Chapter 3) and in-context learning in large language models (Chapter 6).

Active Subset Selection Related to active learning, active subset selection (Mindermann et al., 2022; Evans et al., 2023) aims to interactively construct subsets of the training data to speed up model training or improve model performance. Unlike active learning, active subset selection assumes that the data are fully observed, i.e. both features and labels are known. The acquisition functions used by Mindermann et al. (2022) and Evans et al. (2023) are similar to EPIG in that they attempt to minimize the predictive loss of the model on a set of held-out datapoints.

2.3.2 Priors, Regularization, and Inductive Biases

Active learning controls the data acquisition process assuming a particular choice of model. However, the choice of the model itself, and how we infer its parameters, can significantly affect data-efficiency.

Priors and Regularization In previous sections, we have already discussed the issue of overfitting in deep learning, i.e. the tendency of deep learning to over-optimize on small training datasets. As discussed, Bayesian deep learning can be a helpful tool to guard against overfitting and, for deterministic deep learning, regularized ERM objectives are popular with a similar goal.

Inductive Biases Bayesian inference or regularization objectives can, in principle, be combined with any model architecture. However, we might want to choose (or develop) a model architecture that is suited to the particular domain the model will be applied to. For example, CNNs (see Section 2.2.2) are well-suited to image classification tasks, as both CNN predictions and classification labels do not change (approximately) when the target object is moved across the input image. Given a dataset of limited size, CNNs can achieve better image classification performance than other deep learning architectures such as MLPs, which do not respect the translation invariance of the task, see LeCun et al. (1998) and Murphy (2022, Ch. 14.3.1). Conversely, CNNs perform badly at tasks where outputs depend on the precise location of an object in the input image.

Architecture choices that restrict (in a hard or soft manner) the functions that can be learned by a model are referred to as *inductive biases* (Mitchell, 1980; Murphy, 2023). There exist more precise definitions of the term, but we will stick with the intuitive meaning for the sake of simplicity here. If the inductive bias favours functions with lower risk, it can help machine learning be more data-efficient.

However, when the amount of training data is large, inductive biases that were useful for smaller data can become less important or even harmful. For example, there has recently been a surge of image classification models that do not rely on convolutions as their main building block: Dosovitskiy et al. (2021) apply multi-head self-attention between patches of the input image and Tolstikhin et al. (2021) rely only on MLPs applied to and between embeddings of input patches. While these models still share a significant amount of the operations applied to the individual image patches, translation-invariance is *not* core to their motivation. This might contribute to them requiring much more data to train than CNNs. However, when sufficiently large datasets are available, these models can outperform CNN-based architectures. One can speculate that this is because translation invariance is not exactly the right bias for image classification after all. And, if there is sufficient data and compute available, it may be better to not make restrictive assumptions on the functions the model can learn.

Naive application of deep learning to tabular data has struggled to match the performance of established baselines, especially for small datasets (Popov et al., 2020; Grinsztajn et al., 2022; Borisov et al., 2022; Shwartz-Ziv and Armon, 2022). Perhaps, current deep learning architectures do not have the right inductive biases to perform well on tabular data. In Chapter 3, we will propose a novel deep learning architecture which can outperform established non-deep methods. We achieve this by carefully crafting the inductive bias of the architecture, using blocks of self-attention to attend between individual features and datapoints. We further make careful use of regularization techniques, such as randomly dropping out input features, to avoid overfitting.

2.3.3 Semi-Supervised Learning, Transfer Learning, and In-Context Learning

Next, we discuss a set of approaches for data-efficient machine learning that attempt to make more out of the data that *is* available, rather than collecting additional data or modifying model architectures.

Semi-Supervised Learning Semi-supervised learning starts from a similar scenario as pool-based active learning, assuming a large pool of inputs, some of which have labels available. Unlike active learning, semi-supervised learning tries to extract information from both the labeled and unlabeled datapoints in the pool (Bengio et al., 2006; Hinton et al., 2006; Chapelle et al., 2009; Erhan et al., 2010; Kingma et al., 2014). It further assumes the data are fixed and we cannot acquire additional labels.

Semi-supervised learning often proceeds in a two-step process. First, the unlabeled data are used to learn a lower-dimensional representation in unsupervised fashion. Then, the labeled data are used to train a classifier on top of the representations.

It is possible to use semi-supervised learning techniques to improve data-efficiency for active learning (which typically neglects information in unlabelled datapoints), see e.g. Seung et al. (1992), Muslea et al. (2002), Bickford Smith et al. (2024), and Osband et al. (2023). This is a good example of how approaches to data-efficient deep learning can be combined for maximum effect.

The NPT architecture we introduce in Chapter 3 is naturally suited to semi-supervised learning: the masking mechanism allow NPTs to effortlessly learn from unlabelled datapoints in a semi-supervised setting.

Transfer Learning Transfer learning (Girshick et al., 2014; Donahue et al., 2014; He et al., 2019) proposes a creative solution to a lack of data for the target task. In transfer learning, we bring in data from an *auxiliary* task that is distinct from the target task, but for which a large number of datapoints are available. If the

two tasks are related, the aim is to *transfer* knowledge from the auxiliary to the target task. This often takes the form of a *pre-train–then–finetune* approach: the model is first trained on the auxiliary tasked and then, without re-initializing the model parameters, trained on the target task. The initial training on the auxiliary task is referred to as *pre-training* and the subsequent training on the target task as *finetuning*. If necessary, small architecture changes may be introduced in the finetuning stage to adapt the model to the target task.

Transfer learning is related to semi-supervised learning, as, sometimes, the auxiliary task may be unsupervised while the target task is supervised (Bengio et al., 2006; Hinton et al., 2006). However, unlike semi-supervised learning, transfer learning typically assumes that the task distributions are different between the two stages.

In computer vision, transfer learning became popular with the emergence of large scale datasets such as ImageNet (Deng et al., 2009). In 2012, the ImageNet database contained more than 1 million images (Deng et al., 2012), much larger than many other datasets in computer vision at the time. With transfer learning, performance on computer vision tasks with small datasets (relative to ImageNet) could be improved (Girshick et al., 2014; Donahue et al., 2014; He et al., 2019). In recent years, transfer learning has been applied to ever larger datasets, and state-of-the-art results in computer vision are often achieved via transfer learning (Joulin et al., 2016; Sun et al., 2017; Mahajan et al., 2018; Radford et al., 2021; Kolesnikov et al., 2020).

In natural language processing, a similar shift towards transfer learning took place (Ruder et al., 2019; Howard and Ruder, 2018; Malte and Ratadiya, 2019; Qiu et al., 2020; Li et al., 2020). Perhaps emblematic of this is the BERT language model (Devlin et al., 2019) and its "BERT-like" successors (Lan et al., 2020; Liu et al., 2019; Sanh et al., 2019). These models are trained to reconstruct masked-out words from input sentences (Taylor, 1953) derived from a large dataset of around 4 billion words of natural language. The representations obtained from BERT-like models contain useful information, e.g. about the semantic meaning of a sentence. Finetuning BERT-embeddings on NLP tasks – which often contain

just thousands of examples – led to much improved performance compared to training specialized architectures from scratch. Thus, for a while, transfer learning became the dominant paradigm in NLP.

In-Context Learning Recently, this has been challenged with the emergence of in-context learning (Brown et al., 2020) in large language models (LLMs). Compared to language models from the BERT generation (which are retro-actively sometimes also referred to as LLMs), today's LLMs have up to about 1000 times more parameters and are trained on up to about 1000 times more data (Brown et al., 2020; Chowdhery et al., 2022; Hoffmann et al., 2022; Zhang et al., 2022a). Given their large parameter counts, it is very expensive to finetune these models. One direction of research has thus been to reduce the cost of finetuning, typically by reducing the number of parameters that are finetuned (Houlsby et al., 2019; Hu et al., 2021; Ruder et al., 2022; Li and Liang, 2021).

However, Brown et al. (2020) have shown that, with LLMs, there exists an alternative to finetuning to adapt these models to NLP tasks: in-context learning (ICL). ICL is the ability of LLMs to learn tasks from very few demonstrations *in-context*, i.e. by simply prepending them before the query as part of the input. In other words, instead of using the examples from the task to update the *parameters* of the model, ICL allows LLMs to adapt to novel tasks on the fly, by including a small number of task demonstrations in each input before the relevant test query. In Chapter 6, we carefully characterize the learning of label relationships in ICL and contrast it against expectations from conventional learning algorithms.

This concludes our overview of related work relevant to this thesis. The following chapters of the thesis present our main contributions. In them, we will extend our discussions of related work to include additional references that more closely relate to the topics of the individual chapters.

3

Beyond Individual Input-Output Pairs in Deep Learning

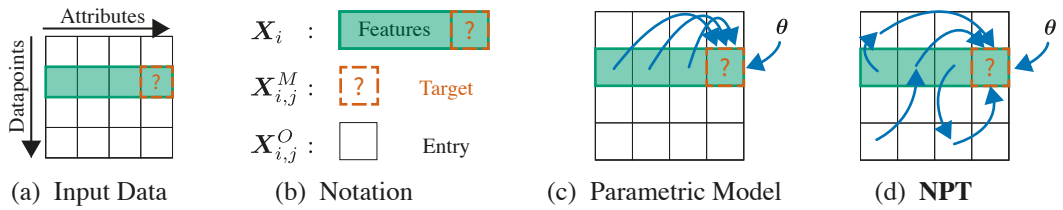
Contents

Authorship Statement

The work presented in this chapter has previously been published as Kossen*, Band*, et al. (2021). Section 1.3 gives a detailed account of the contributions my collaborators have made to this work.

3.1 Introduction

As we have previously discussed in Section 2.2, from CNNs (LeCun et al., 1998) to Transformers (Vaswani et al., 2017), most of supervised deep learning relies on *parametric* modeling. And, as previously discussed in Section 2.1.2, supervised parametric models learn parameters $\boldsymbol{\theta}$ from a set of training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ to maximize training likelihoods $p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$ mapping from features $\boldsymbol{x} \in \mathcal{X}$ to target values $y \in \mathcal{Y}$. At test time, they then make a prediction $p(y^* \mid \boldsymbol{x}^*; \boldsymbol{\theta})$ that depends only on those parameters $\boldsymbol{\theta}$ and the test input $\boldsymbol{x}^*$. That is, parametric models do not consider direct dependencies between datapoints.



(a) Input Data (b) Notation (c) Parametric Model (d) **NPT**

Figure 3.1: NPTs directly learn interactions between datapoints. (a) Input data: predict masked target entry [?] for datapoint $\boldsymbol{X}_i$. (b) Notation from Section 3.2. (c) Parametric models predict only from the features of the given input. (d) NPTs predict by modeling relationships between all points in the dataset.

In this chapter, we present a data-efficient deep learning architecture for tabular data that we call Non-Parametric Transformers (NPTs).[1] With NPTs, we challenge parametric modeling as the dominant paradigm in deep learning. Based on the same end-to-end learning motivations that underpin deep learning itself, we consider giving models the *additional flexibility* of using training data *directly* when making predictions, $p(y^* \mid \boldsymbol{x}^*, \mathcal{D}_{\text{train}}; \boldsymbol{\theta})$.

NPTs are a general deep learning architecture that takes the entire dataset as input and predicts by explicitly *learning* interactions between datapoints, see Fig. 3.1 for an overview. NPTs leverage both parametric and *non*-parametric predictive mechanisms, with the use of end-to-end training allowing the model to naturally learn from the data how to balance the two. Namely, instead of just learning predictive functions from the features to the targets of independent datapoints, NPTs can also learn to reason about general relationships *between* inputs. We use multi-head self-attention (Bahdanau et al., 2015; Vaswani et al., 2017; Lee et al., 2019) to model relationships between datapoints and construct a training objective for NPTs with a stochastic masking mechanism inspired by self-supervised reconstruction tasks in natural language processing (Devlin et al., 2019). We show that NPTs *learn* to look up information from other datapoints and capture the causal mechanism generating the data in semi-synthetic settings. However, unlike conventional non-parametric models, such as those discussed in Section 2.1.3, NPTs are not forced to *only* make predictions in this way: they can also make use of ordinary parametric deep learning.

[1] Code for NPTs is available at github.com/OATML/Non-Parametric-Transformers.

While questioning parametric modeling assumptions is unconventional in deep learning, we have already introduced non-parametric models in Section 2.1.3 as a popular and well-known alternative in traditional machine learning and statistics. We have previously highlighted how non-parametric models, such as those based on kernels, make predictions in explicit dependence on the training data $p(y^* \mid \boldsymbol{x}^*, \mathcal{D}_{\text{train}})$, typically do not have any explicit parameters, and interpolate between training points according to a fixed procedure. The interactions between inputs are fully defined by architectural choices and a small set of hyperparameters that must be carefully chosen. Conventional non-parametric models cannot *learn* – in the sense familiar to deep learning practitioners – interactions from the data, limiting the flexibility these models have in adapting to the data at hand.

Approaches such as Deep Gaussian Processes (Damianou and Lawrence, 2013), Deep Kernel Learning (Wilson et al., 2016), and Neural Processes (Garnelo et al., 2018b; Garnelo et al., 2018a; Kim et al., 2019) have all sought to apply ideas from deep neural networks to non-parametrics. Compared to NPTs, these approaches rely heavily on motivations from stochastic processes. This leads to them being either less flexible than NPTs or requiring strong assumptions on the data, making them inapplicable to the practical scenarios considered in this chapter. Unlike previous work, NPTs explicitly learn to predict from interactions between datapoints, and they can be applied to general supervised machine learning tasks. We refer to Section 3.3 for an overview of these and other related approaches.

A key contribution of this method is opening the door to a more general treatment of how deep learning models can make use of dependencies between datapoints for predictions. Our results demonstrate that NPTs make use of interactions between datapoints in practice. We further show highly competitive performance on several established tabular datasets. This is significant, as tabular datasets are often small and deep learning methods have struggled to match the performance of established baselines on these tasks. Additionally, we show that NPTs can solve complex reasoning tasks by combining representation learning and cross-datapoint lookup;

something that is impossible for conventional deep learning or non-parametric models due to their inability to *learn* relations between datapoints.

3.2 The NPT Architecture

NPTs explicitly learn relationships between datapoints to improve predictions. To accomplish this, they rely on three main ingredients: **(1)** We provide the model with the entire dataset – all datapoints – as input. We approximate this with mini-batches where necessary for large data. At test time, both training and test data are input to the model; during training, the model learns to predict targets from the training data (Section 3.2.6). **(2)** We use self-attention between datapoints to explicitly model relationships between datapoints. For example, at test time, the attention mechanism models relationships amongst training points, amongst test points, and between the two. **(3)** NPT's training objective is to reconstruct a corrupted version of the input dataset. Similar to the BERT Transformer model (Devlin et al., 2019) (briefly discussed in Section 2.2.2), we apply stochastic masking to inputs and minimize a loss on predictions at entries masked out in the input. Next, we introduce the three components in detail.

3.2.1 Datasets as Inputs

NPTs take the entire dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ as input. The datapoints are stacked as the rows of this matrix $\{\boldsymbol{X}_{i,:} \in \mathbb{R}^d \mid i \in 1 \dots n\}$, and we refer to the columns as attributes $\{\boldsymbol{X}_{:,j} \in \mathbb{R}^n \mid j \in 1 \dots d\}$. Each attribute is assumed to share a semantic meaning among all datapoints. In single-target classification and regression, we assume that the targets (labels) are the final attribute $\boldsymbol{X}_{:,d}$, and the other attributes $\{\boldsymbol{X}_{:,j} \mid j \neq d\}$ are input features, e.g. the pixels of an image. Each $\boldsymbol{X}_{i,j}$ is an entry or value. In addition to tabular data, many modalities such as images, graphs, or time series can be reshaped to fit this format. Note that this is a departure from common notation for supervised learning as introduced in Section 2.1, as the input $\boldsymbol{X}$ now includes both features and targets (collectively, attributes). We also now use capital letters for *matrices* instead of random variables.

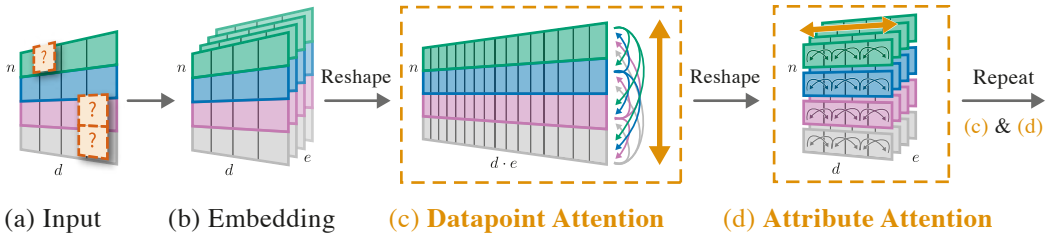


Figure 3.2: Overview of the Non-Parametric Transformer. (a) The input dataset and mask matrix are stacked and (b) linearly embedded for all datapoints independently. NPT then applies (c) Attention Between Datapoints (ABD, Section 3.2.4) across all n samples of hidden dimension $h = d \cdot e$. (d) Attention Between Attributes (ABA, Section 3.2.5) then attends between the attributes for each datapoint independently. We repeat steps (c) and (d) and obtain a final prediction from a separate linear projection (not shown).

In masked language modeling (Devlin et al., 2019), mask tokens denote which words in a sentence are unknown and where, at training time, model predictions will have a loss backpropagated. Analogously, we use a binary matrix $\boldsymbol{M} \in \mathbb{R}^{n \times d}$ to specify which entries are *masked* in the input $\boldsymbol{X}$. This matrix is also passed to NPTs as input. The task is to predict the masked values $\boldsymbol{X}^M = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 1\}$ from the observed values $\boldsymbol{X}^O = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 0\}$. More precisely, NPTs predict the distribution over masked values given observed values, $p(\boldsymbol{X}^M \mid \boldsymbol{X}^O; \boldsymbol{\theta})$.

In summary, NPTs take as input the entire dataset and masking matrix $(\boldsymbol{X}, \boldsymbol{M})$, and make predictions for values masked at input. This general setup accommodates many machine learning settings simply by adjusting the placement of the binary masks in $\boldsymbol{M}$. We focus on single-target classification and regression but outline multi-target settings, imputation, self-supervision using input features, and semi-supervision in Appendix A.3.4. Next, we describe the NPT architecture.

3.2.2 NPT Architecture

An overview of the Non-Parametric Transformer (NPT) is depicted in Fig. 3.2. NPTs receive the dataset and masking matrix $(\boldsymbol{X}, \boldsymbol{M})$ as input (Fig. 3.2a). We stack these and apply an identical linear embedding to each of the n datapoints, obtaining an input representation $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}$ (Fig. 3.2b). Next, we apply a sequence of multi-head self-attention layers (Vaswani et al., 2017; Devlin et al.,

2019; Bahdanau et al., 2015). Crucially, we alternatingly apply attention between *datapoints* and attention between *attributes* of individual datapoints (Figs. 3.2c-d).

These operations allow our model to learn both relationships between datapoints as well as transformations of individual datapoints. Finally, an output embedding gives the prediction $\hat{\boldsymbol{X}} \in \mathbb{R}^{n \times d}$, which now has predicted values at entries that were masked at input. We refer to Appendix A.3.3 for details, such as treatment of categorical and continuous variables. Importantly:

Property 1. *NPTs are equivariant to a permutation of the datapoints. (Cf. Appendix A.1 for proof.)*

In other words, if the set of input datapoints is shuffled, NPTs produce the same prediction but shuffled in an analogous manner. This explicitly encodes the assumption of exchangeability discussed in Section 2.1: the learned relations between datapoints should not depend on their ordering. At a high level, permutation-equivariance holds because all components of NPTs are permutation-equivariant, and the composition of permutation-equivariant functions is itself permutation-equivariant. We now briefly recap multi-head self-attention which plays an important role throughout the NPT architecture.

3.2.3 Multi-Head Self-Attention

Multi-head self-attention (MHSA) is a powerful mechanism for learning complex interactions between elements in an input sequence. Popularized in natural language processing (Vaswani et al., 2017; Bahdanau et al., 2015), MHSA-based models have since been successfully applied to many areas of machine learning, see Sections 2.2.2 and 3.3.

Dot-product attention computes attention weights by comparing queries $\{\boldsymbol{Q}_i \in \mathbb{R}^{1 \times h_k} \mid i \in 1 \dots n\}$ with keys $\{\boldsymbol{K}_i \in \mathbb{R}^{1 \times h_k} \mid i \in 1 \dots m\}$, ultimately updating the representation of the queries by aggregating over values $\{\boldsymbol{V}_i \in \mathbb{R}^{1 \times h_v} \mid i \in 1 \dots m\}$ via the attention weights. We stack the queries, keys, and values into matrices

$\boldsymbol{Q} \in \mathbb{R}^{n \times h_k}$, $\boldsymbol{K} \in \mathbb{R}^{m \times h_k}$, and $\boldsymbol{V} \in \mathbb{R}^{m \times h_v}$ and, as is commonly done for convenience, assume $h_k = h_v = h$. Then, we compute dot-product attention as the matrix product

$$\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{h})\boldsymbol{V}, \tag{3.1}$$

Multi-head dot-product attention concatenates a series of k independent *attention heads*

$$\text{MHAtt}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \underset{\text{axis}=h}{\text{concat}}(\boldsymbol{O}_1, \ldots, \boldsymbol{O}_k)\boldsymbol{W}^O, \text{ where} \tag{3.2}$$

$$\boldsymbol{O}_j = \text{Att}(\boldsymbol{Q}\boldsymbol{W}_j^Q, \boldsymbol{K}\boldsymbol{W}_j^K, \boldsymbol{V}\boldsymbol{W}_j^V). \tag{3.3}$$

We learn embedding matrices $\boldsymbol{W}_j^Q, \boldsymbol{W}_j^K, \boldsymbol{W}_j^V \in \mathbb{R}^{h \times h/k}, j \in \{1, \ldots, k\}$ for each head j, and $\boldsymbol{W}^O \in \mathbb{R}^{h \times h}$ mixes outputs from different heads. Here, we focus on multi-head *self*-attention,

$$\text{MHSelfAtt}(\boldsymbol{H}) = \text{MHAtt}(\boldsymbol{Q} = \boldsymbol{H}, \boldsymbol{K} = \boldsymbol{H}, \boldsymbol{V} = \boldsymbol{H}), \tag{3.4}$$

which uses the *same* inputs for queries, keys, and values. Following Transformer best practices to improve performance (Vaswani et al., 2017; Devlin et al., 2019; Lee et al., 2019; Chen et al., 2018; Narang et al., 2021), we first add a residual branch and apply Layer Normalization (LN) (Ba et al., 2016) followed by MHSelfAtt($\cdot$),

$$\text{Res}(\boldsymbol{H}) = \boldsymbol{H}\boldsymbol{W}^{\text{res}} + \text{MHSelfAtt}(\text{LN}(\boldsymbol{H})), \tag{3.5}$$

with learnable weight matrix $\boldsymbol{W}^{\text{res}} \in \mathbb{R}^{h \times h}$. Then, we add another residual branch with LN and a row-wise feed-forward network (rFF), finally giving the full multi-head self-attention layer as

$$\text{MHSA}(\boldsymbol{H}) = \text{Res}(\boldsymbol{H}) + \text{rFF}(\text{LN}(\text{Res}(\boldsymbol{H})) \in \mathbb{R}^{n \times h}. \tag{3.6}$$

3.2.4 Attention Between Datapoints (ABD)

The NPT architecture alternates between two different layers based on MHSA: attention between datapoints (ABD) and attention between attributes (ABA). We first present the ABD layer, which implements the key operation that allows NPTs to reason about the pairwise relationships between all datapoints, see Fig. 3.2c. As

input to ABD, we flatten the output of the previous layer $\boldsymbol{H}^{(\ell)}$ from $\mathbb{R}^{n\times d\times e}$ to $\mathbb{R}^{n\times h}$ with $h = d \cdot e$. Then, we apply MHSA$(\cdot)$ between the intermediate datapoint representations $\{\boldsymbol{H}_i^{(\ell)} \in \mathbb{R}^{1\times h} \mid i \in 1 \dots n\}$ as

$$\text{ABD}(\boldsymbol{H}^{(\ell)}) = \text{MHSA}(\boldsymbol{H}^{(\ell)}) = \boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n\times h}. \tag{3.7}$$

Here, the rFF of each ABD layer is an MLP that is applied independently to each of the n datapoints. At the first ABD layer, we input $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n\times d\times e}$, the linearly embedded input data. After applying ABD, we reshape the output again, from $\mathbb{R}^{n\times h}$ to $\mathbb{R}^{n\times d\times e}$.

Note that this is distinct from how MHSA$(\cdot)$ is usually applied in the literature, as we compute attention between *different datapoints* and not between the *features of a single datapoint*, as done by e.g. Vaswani et al. (2017), Devlin et al. (2019), Dosovitskiy et al. (2021), and Jaegle et al. (2021). For example, in natural language processing, attention is usually applied between the tokens (attributes) of an input datapoint but not between different datapoints. For example, NPTs could learn to attend between two datapoints with indices i and i' by embedding $\boldsymbol{Q}_i$ and $\boldsymbol{K}_{i'}$ in close proximity. Following Eq. (3.1), datapoint i will then attend more closely to i' because $\boldsymbol{Q}_i\boldsymbol{K}_{i'}^T$ will be large. By stacking many ABD layers, NPTs can learn higher-order interactions between datapoints (Vaswani et al., 2017).

3.2.5 Attention Between Attributes (ABA)

We now introduce the attention between attributes (ABA) layer, which typically follows the ABD layer in the NPT architecture. ABA layers can help the model learn better per-datapoint representations, see Fig. 3.2d. For ABA, we apply MHSA$(\cdot)$ independently to each row (corresponding to a single datapoint) in the input $\boldsymbol{H}_i^{(\ell)} \in \mathbb{R}^{d\times e}$, $i \in \{1, \dots, n\}$, giving

$$\text{ABA}(\boldsymbol{H}^{(\ell)}) = \underset{\text{axis}=n}{\text{stack}}(\text{MHSA}(\boldsymbol{H}_1^{(\ell)}), \dots, \text{MHSA}(\boldsymbol{H}_n^{(\ell)})) = \boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n\times d\times e}. \tag{3.8}$$

Just like in standard Transformers, ABA is used to transform attribute representations of single datapoints independently. We batch over the n dimension to compute ABA efficiently.

By alternating between attention between datapoints (ABD) and attributes (ABA), NPTs can model both complex dependencies between points as well as learn suitable transformations of datapoints individually. Next, we describe the use of masking mechanisms during NPT training and evaluation.

3.2.6 Masking and Optimization

Masking Much like in masked language modeling (Devlin et al., 2019), we use masks to indicate which values NPTs are expected to predict, and to prevent the model from accessing ground truth values. Recall that NPTs need to predict $p(\boldsymbol{X}^M \mid \boldsymbol{X}^O)$, with masked values $\boldsymbol{X}^M = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 1\}$ and observed values $\boldsymbol{X}^O = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 0\}$. Masked values can be either features or targets. Canonically, masked language modeling is used to perform self-supervised learning on a sequence of tokens in a sentence (Devlin et al., 2019). We use such *stochastic feature masking* to mask feature values $\boldsymbol{X}_{i,j}, j \neq d$, with probability p_{feature} during training. We also apply stochastic masking to the targets of the training set $\boldsymbol{X}_{:,d}$ with probability p_{target}. We call this *stochastic target masking*. Note that we take great care to avoid test set leakage and *never* reveal targets of the test set to the NPT. We refer to Appendix A.3.4 for full details of our masking procedure in a variety of settings.

NPT Objective During training, we compute the negative log-likelihood loss at training targets $\mathcal{L}^{\text{Targets}}$ as well as the auxiliary loss from masked-out features $\mathcal{L}^{\text{Features}}$. We write the NPT training objective as

$$\mathcal{L}^{\text{NPT}} = (1 - \lambda)\mathcal{L}^{\text{Targets}} + \lambda\mathcal{L}^{\text{Features}}, \tag{3.9}$$

where λ is a hyperparameter. At test time, we only mask and compute a loss over the targets of test points. We give full optimization details in Appendix A.3.5.

The stochastic feature masking loss $\mathcal{L}^{\text{Features}}$ requires NPTs to make predictions over all attributes, encouraging the models to learn a representation of the entire dataset. This increases the difficulty of the task and adds more supervision, which we find tends to have a beneficial regularizing effect. This is important, in particular, because many of the tabular datasets on which we evaluate NPTs are small.

Interestingly, stochastic *target* masking means that many training targets are *unmasked* to the model at training time. This allows NPTs to learn to predict the masked targets of certain training datapoints using the *targets of other training datapoints* in addition to all input features.[2]

NPTs no longer have to memorize a mapping between training inputs and outputs in their parameters $\boldsymbol{\theta}$, and can instead use their representational capacity to learn functions using other *training features and targets as input*. For example, NPTs could learn to assign test datapoints to clusters of training datapoints, and predict on those points using interpolation of the training targets in their respective cluster. We explore the ability of NPTs to solve such tasks in Section 3.4.2.

Handling Large Datasets Due to the poor $\mathcal{O}(n^2)$ time and space complexity of self-attention, we resort to approximations once the data grows too large. For example, we reach 24 GB of GPU memory for standard model and data feature sizes at about 8000 datapoints. We find that processing the data in random subsets for model training and prediction, i.e. *mini-batching*, is a simple and effective solution. We construct mini-batches such that, at test time, training and test data are both present in the same batch, to allow NPTs to attend to training datapoints. In Section 3.4.3, we show that NPTs make use of attention between datapoints with mini-batching enabled.

3.3 Related Work

Deep Non-Parametric Models Deep Gaussian Processes (Damianou and Lawrence, 2013) and Deep Kernel Learning (DKL) (Wilson et al., 2016) extend ideas from Gaussian Processes (Rasmussen, 2003) to representation learning. Deep GPs stack standard GPs with the aim to learn more expressive relationships between input points, sharing motivation with NPTs. However, unlike NPTs, deep GPs

[2] A concern here could be that the model will memorize training targets and fail to generalize. In practice, we do not observe generalization issues, likely because (i) a loss is never backpropagated on an unmasked value, and (ii) BERT-style masking (Devlin et al., 2019) uses token randomization to prevent memorization. See Appendix A.3.4.

are difficult to work with in practice, requiring complex approximate inference schemes (Dai et al., 2016; Bui et al., 2016; Salimbeni and Deisenroth, 2017). DKL applies a neural network to each datapoint *independently* before passing points on to a standard Gaussian Process, making predictions based directly on similarity in embedding space instead of *learning* the interactions themselves.

Neural Processes We here extend on our previous discussion of Neural Processes (NPs) in Section 2.2.2 and relate them to the NPT architecture. Like NPTs, NPs also make predictions in direct dependence on all input datapoints, implementing a deep non-parametric prediction mechanism which they learn from the data. However, NPTs and NPs have different training regimes which is why we cannot compare the two: while we train NPTs on individual datasets, NPs require a large collection of datasets to be trained on (Kim et al., 2019). Note that, in principle, there is nothing preventing us from training NPTs over a collection of dataset, and we see such "neural process NPTs" as exciting future work.

Attention As discussed in Section 2.2.2, NPTs are part of a line of recent work that explores the use of Transformer-based architectures outside of natural language processing, e.g. Transformers in computer vision (Parmar et al., 2018; Dosovitskiy et al., 2021; Jaegle et al., 2021) or architectures exploiting desirable invariances or equivariances (Lee et al., 2019; Locatello et al., 2020; Fuchs et al., 2020; Hutchinson et al., 2021). Like NPTs, Set Transformer (Lee et al., 2019) attends to a set of input points. However, unlike NPTs, Set Transformer relies on the existence of multiple independent sets for training and makes only a single prediction for each set. Like NPTs, Axial Transformers (Ho et al., 2019) and MSA Transformers (Rao et al., 2021) attend to multiple dimensions of matrix-shaped input. However, Axial Transformers process single images as input, i.e. no attention across datapoints is performed. MSA Transformers use attention within individual protein sequences and across an aligned protein family for contact prediction, but do not consider a more general setting. Recent works have improved neural network performance on tabular data using attention. AutoInt (Song et al., 2019) is a direct application

of multi-head attention to tabular data, and TabNet (Arik and Pfister, 2021) sequentially attends to sparse subsets of the features inspired by tree-based models. Both approaches do not reason about interactions between datapoints, which is a key feature of NPTs. Concurrently to our original publication, Somepalli et al. (2021) have proposed an architecture similar to NPTs. While they also attend between datapoints and attributes, their SAINT model requires separate contrastive pre-training and finetuning phases, and they do not stack attention between datapoints across multiple layers, limiting expressivity.

Semi-Supervised Learning and Graph Neural Networks NPTs also relate to semi-supervised learning (cf. Section 2.3) and transductive learning (Vapnik, 2006), which both make use of unlabeled inputs during training. NPTs natively support both settings by simply including any unlabeled datapoints in the input matrix at training time with masked-out targets. The body of related work includes semi-supervised and transductive learning on graphs with graph neural networks (GNNs), e.g. Kipf and Welling (2017), Garcia and Bruna (2018), Veličković et al. (2018), Kipf et al. (2018), and Xu et al. (2019). NPTs can be seen as a generalization of GNNs in which a set of dependencies (edges) between datapoints is not known a priori and is instead learned from data using self-attention. Like NPTs, Neural Relational Inference (NRI) (Kipf et al., 2018) attempts to discover relations amongst datapoints. However, NRI lacks scalability because it requires that embeddings be stored for each potential graph edge.

Meta-Learning and In-Context Learning In Section 3.4.2, we apply NPTs to tasks that require learning of relational structure between datapoints on training data to achieve good generalization performance on novel test inputs. This setup shares motivations with meta-learning (Biggs, 1985; Bengio et al., 1991; Lake et al., 2015; Finn et al., 2017), in which a model is *pre-trained* on a variety of tasks, such that it can then learn new tasks using only a small number of additional training points from the new task. However, we consider evaluation without any additional gradient updates, unlike recent meta-learning methods (Finn

et al., 2017; Yoon et al., 2018) which are therefore inapplicable to this setting. This further relates to in-context learning (ICL) in LLMs (Brown et al., 2020) (cf. Section 2.3 and Chapter 6): similar to ICL, we consider attention to a "context" set of datapoints. However, NPTs are trained on a single task and insensitive to the order in which the context datapoints are provided, unlike ICL (Lu et al., 2022). We expand on this discussion in Chapter 7.

Metric Learning (Deep) Metric Learning aims to learn distance functions such that the (semantic) similarity and dissimilarity between input points is meaningfully captured, e.g. Vijayakumar and Schaal (1997), Roweis et al. (2004), Vinyals et al. (2016), Movshovitz-Attias et al. (2017), Wang et al. (2019b), and Seidenschwarz et al. (2021). Similarly, retrieval models in NLP learn to look up relevant training instances for prediction (Guu et al., 2018; Hashimoto et al., 2018; Guu et al., 2020). The attention between datapoints in NPTs can be seen as implicitly learning exactly such (dis-)similarity. Usually, metric learning embeds inputs by applying the same embedding function independently to each datapoint. This is in contrast to NPTs, which leverage a learned self-attention mechanism between test inputs and training datapoints (including their labels) at prediction time.

3.4 Experiments

We seek to answer the following set of questions in our evaluation of NPTs: (**Q1**) How do NPTs perform on standard benchmarks for supervised machine learning? (**Q2**) Can NPTs successfully model interactions between datapoints in idealized settings? (**Q3**) Do NPTs actually learn to rely on interactions between datapoints for prediction on real-world datasets? (**Q4**) If so, what is the nature of these interactions, e.g. which other datapoints are relevant for prediction?

3.4.1 NPTs Perform Competitively on Established Benchmarks

To answer **(Q1)**, we evaluate NPTs on tabular data from the UCI Repository (Dua and Graff, 2017). Tabular data is ubiquitous in real-world machine learning (Chui et al., 2018) but notoriously challenging for general purpose deep neural networks, which are rarely used in practice here because they are consistently outperformed by boosting models (Schapire, 1990).[3]

Tabular Datasets, Setup, and Baselines We evaluate NPTs over ten datasets varying across the number of datapoints, number of features, composition (categorical or continuous) of features, and task. Four of the ten are binary classification, two are multi-class classification, and four are regression. We compare NPTs against a wide set of standard or state-of-the-art baselines: Random Forests (Breiman, 2001), Gradient Boosting Trees (Friedman, 2001), XGBoost (Chen and Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), LightGBM (Ke et al., 2017), MLPs, k-NN (Fix, 1985; Altman, 1992), and TabNet (Arik and Pfister, 2021). We tune the parameters of all models on validation sets and use 10-fold cross-validation whenever computationally feasible. Note that while we perform an extensive grid search for the baselines, we only search over a small set of configurations for NPTs. We refer the reader to Appendix A.4 for further details on the setup for datasets and baselines, and Appendix A.3.1 for NPT hyperparameters.

Tabular Data Results We report the average rank order for NPTs and various tree-based and deep learning baselines in Table 3.1. NPTs achieve the highest average ranking on binary and multi-class classification tasks, outperforming CatBoost and XGBoost, two popular state-of-the-art boosting methods designed specifically for tabular data. On regression tasks, NPTs tie in average rank with XGBoost, and are outperformed only by CatBoost. In addition to their strong rank-wise performance,

[3]We conduct an informal survey of all Kaggle (Inc., 2021) competitions using tabular data completed in 2020 with a public leaderboard. In 11 out of a total of 13 cases, the winning entries relied on some form of boosting.

Table 3.1: Average rank order of various methods (± standard error) on UCI benchmarks, across binary classification, multi-class classification, and regression tasks. We determine rank using the test area under the receiver operating characteristic (AUROC) curve on binary classification (4 of 10 datasets), accuracy on multi-class classification (2 of 10), and root mean squared error (RMSE) on regression (4 of 10), and sort methods by ascending rank for each metric. See Appendix A.2.6 for the full results.

Method	AUROC
NPT	**2.50 ± 0.87**
CatBoost	2.75 ± 0.85
LightGBM	3.50 ± 1.55
XGBoost	4.75 ± 1.25
Gradient Boosting	5.00 ± 0.71
MLP	5.75 ± 1.49
Random Forest	6.00 ± 0.71
TabNet	6.50 ± 1.32
k-NN	8.25 ± 0.48

Method	Accuracy
NPT	**2.50 ± 0.50**
XGBoost	**2.50 ± 1.50**
MLP	3.00 ± 2.00
CatBoost	3.50 ± 0.50
Gradient Boosting	3.50 ± 1.50
Random Forest	6.50 ± 0.50
TabNet	7.50 ± 0.50
LightGBM	7.50 ± 1.50
k-NN	8.50 ± 0.50

Method	RMSE
CatBoost	**3.00 ± 0.91**
XGBoost	3.25 ± 0.63
NPT	3.25 ± 1.31
Gradient Boosting	4.00 ± 1.08
Random Forest	4.50 ± 0.87
MLP	5.00 ± 1.22
LightGBM	6.50 ± 1.55
TabNet	6.75 ± 0.95
k-NN	8.75 ± 0.25

NPTs achieve the best performance on 4 out of the 10 benchmark datasets – more than any other method. These are remarkable results for a general purpose model that does not include tabular-specific design, supporting our hypothesis that attention between datapoints is a useful architectural inductive bias for prediction. For all metrics across all datasets, i.e. NLL for classification, AUROC/accuracy for binary/multi-class classification, and (R)MSE for regression, we refer the reader to Appendix A.2.6. In the appendix, we present ablations which suggest that the performance of NPTs is robust across a wide range of hyperparameter choices (Appendix A.2.4) and that both the introduction of the ABA layer and the stochastic feature masking contribute positively to the performance of NPTs (Appendix A.2.5).

3.4.2 NPTs Can Learn to Predict Using Attention Between Datapoints

To determine if NPTs can successfully learn to exploit interactions between datapoints **(Q2)**, we introduce a task with strong input correlations for which we know ground-truth interactions. Concretely, we use the UCI Protein regression dataset (cf. Section 3.4.1) to construct the following semi-synthetic task: for each batch, we input the original data with masked target values as well as a *copy* of the original data where all target values have been revealed, i.e. no masking is applied

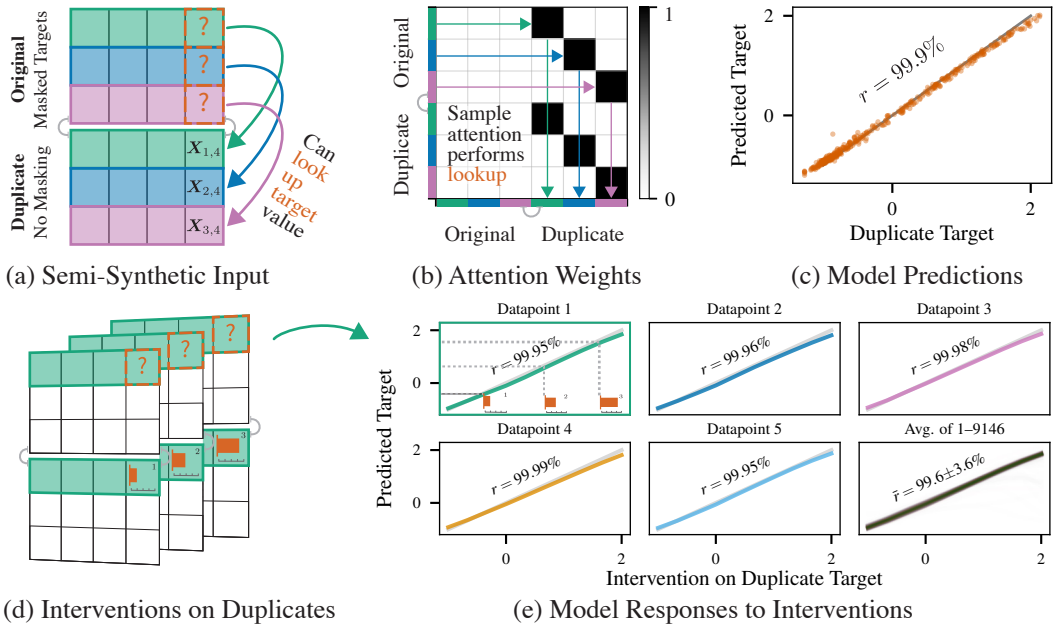


Figure 3.3: Demonstrating NPTs' ability to predict from Attention Between Datapoints (ABD). (a) We append to the original data with masked targets [?] a copy of the same data with all masked values revealed, such that perfect prediction via lookup is possible. (b) Attention weights indicate that the ideal lookup behavior is learned by the NPT. Shown are actual values learned by the NPT at head 0 and depth 4 for the first 3 datapoints. (c) NPT predictions closely match the ideal values. (d) Additionally, we intervene on the values of individual targets, (e) finding that NPT predictions adjust accordingly.

(Fig. 3.3a). NPTs can use attention between datapoints to achieve arbitrarily good performance by *learning* to look up the target values in the matching duplicate row. At test time, we input novel semi-synthetic test data to ensure that the NPT has learned the correct relational mechanism and not just memorized target values.

NPTs successfully learn to perform this lookup between original and duplicate datapoints. The ABD attention weights, visualized for the first three datapoints in Fig. 3.3b, clearly show the model correctly attending to the duplicates. As a result, NPT predictions are Pearson-correlated with the duplicate targets at $r = 99.9\%$ (Fig. 3.3c). This equals an RMSE of only 0.44, much lower than the error on the original Protein dataset (Table A.7).

Purely parametric models cannot exploit information from other datapoints, limiting their performance on this task. Non-parametric approaches also cannot solve this task in its original form, because unlike NPTs they must be told which datapoints are the originals (training data) and which the duplicates (test data).

We demonstrate in Appendix A.2.1 that even when we make these concessions, we can easily adapt the task such that both k-Nearest Neighbors and Deep Kernel Learning fail to solve it. In fact, we are not aware of any other model that can solve the adapted task.

Additionally, we perform an *interventional* experiment to investigate the extent to which NPTs have actually learned the causal mechanism underlying the lookup task. As illustrated in Fig. 3.3d, we now intervene on individual duplicate datapoints at test time by varying their target value across a wide range. We stress that we perform these experiments without retraining the model, using exactly the same NPT from Figs. 3.3a-c. The model is now confronted with target values associated with features that are highly unlikely under the training data. This label distribution shift (Garg et al., 2020) is a challenging setting for neural networks. However, NPT predictions follow the intervened target values with near-perfect correlation, Fig. 3.3e, continuing to predict by correctly looking up targets.

We now confidently conclude that NPTs robustly learn the causal data-generating mechanism underlying the semi-synthetic dataset. This requires NPTs to *learn* a non-trivial sequence of computational steps. They must learn to match rows based on similarity of relevant features; to look up the target value of the duplicated datapoint; and, to copy that value into the target of the masked datapoint.

3.4.3 NPTs Learn to Use Attention Between Datapoints on Real Data

We next consider **(Q3)**: do NPTs actually learn to use attention between datapoints for prediction on real data? We design a test that allows us to quantify the extent to which the predictions of an NPT trained in standard fashion on one of our benchmark datasets depend on relationships between datapoints at test time. Concretely, for each target value in the input we randomize the data for all *other* datapoints by independently shuffling each of their attributes across the rows. We then evaluate the loss on the prediction at the target entry and repeat this procedure for all test datapoints. This completely corrupts the information from all datapoints except

Table 3.2: Drop in NPT performance after destroying information from other datapoints. Shown are changes in test set performance, where negative values indicate worse performance after corruption.

Δ *Accuracy*	Poker	Income	Higgs	Forest	Kick	Breast Cancer
	−1.1	−1.1	−0.5	−0.1	−0.1	0.0
$\Delta_{RMSE}/_{RMSE}$ (%)	Yacht	Protein	Boston	Concrete		
	−52%	−21%	−20%	−7%		

the one for which we evaluate, while preserving high-level statistics of the input. Hence, a model that relies meaningfully on attention between datapoints will show deteriorating performance. We give an algorithm for the corruption procedure as well as further discussion in Appendix A.2.2.

We report the resulting change in performance after corruption in Table 3.2 for all datasets from Section 3.4.1. We find that for most datasets, the corruption of other rows at test time significantly decreases the performance of the trained NPT models. This indicates that the NPTs have successfully learned to make predictions supported by attention between datapoints. For some datasets, the corruption experiment deteriorates performance completely. For example, for the Protein regression dataset NPTs achieve state-of-the-art performance, but corrupting the input at test time leads to NPTs performing worse than all of the baselines considered in Section 3.4.1. We note that minor differences in performance are often still significant, as differences between competing models in Section 3.4.1 are often likewise small.

Interestingly, on certain datasets such as Forest Cover, Kick, and Breast Cancer, corrupted inputs do not significantly affect performance. It appears that when NPTs do not find it advantageous to rely on attention between datapoints during training, they can learn to completely ignore other inputs, essentially collapsing into a standard parametric model. This supports our earlier claims that NPTs can learn end-to-end from data the extent to which they rely on other datapoints for prediction. We think this is extremely interesting behavior and are unaware of prior work reporting similar results. However, we stress that these results reflect

inductive biases of the NPT architecture and do not lend themselves to general statements about the performance of parametric versus non-parametric models.

3.4.4 NPTs Rely on Similar Datapoints for Predictions on Real Data

So far, we have presented evidence that NPTs (sometimes strongly) depend on attention between datapoints. However, we do not know what kind of interactions are learned in practice on real data (**Q4**). As an initial step towards understanding this, we now present two experiments investigating *to which* other datapoints NPTs attend.

Qualitative Evidence Figure 3.4 shows an attention map for attention between datapoints (ABD) of an NPT on a batch of the Protein regression dataset. We sort the input data with respect to their input space distance such that similar datapoints are now close to each other. The diagonal pattern in Fig. 3.4 indicates that NPTs attend more strongly to datapoints that are similar in feature space. Appendix A.2.3 shows additional attention maps and discusses this further.

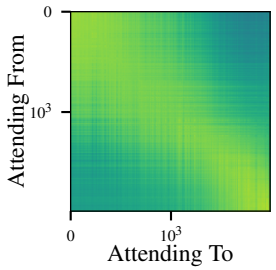


Fig. 4: Attention weights.

Quantitative Evidence Seeking a quantitative measure for this hypothesis, the *data deletion* experiment repeats the following procedure for all test set points: iteratively delete other datapoints from the input if they do not significantly affect the prediction. We stop if less than 2% of the original datapoints remain, or if the total change in prediction for the target (relative to the original prediction with all data) exceeds 10%. We investigate the average input feature space distances between the test point and the *kept* datapoints, as well as the distances between the test point and the *deleted* datapoints. "Input features" here refer to all attributes of the input datapoints that are not labels.

We find that kept datapoints have a significantly lower average feature space distance to the test point than those deleted. This indicates that two datapoints

i, i' that are similar in input feature space, such that $\sum_{j<d}(X_{i,j} - X_{i',j})^2$ is low, have a larger effect on the predictions of one another. A Wilcoxon signed-rank test is significant at $p \approx 8.77 \cdot 10^{-130}$. We give full details on this in Appendix A.2.3.

Both experiments support the hypothesis that NPTs rely on similar datapoints for prediction in real data settings. One possible explanation is that similar datapoints might have different realizations of observation noise which NPTs could learn to average out. Altogether, we conclude that NPTs can and do learn representations which rely on interactions between datapoints for prediction.

3.5 Discussion

IID Data Circling back to our discussion in Section 2.1, NPTs do not assume that the data are IID because their prediction of the masked values given the observed values, $p(\boldsymbol{X}^M \mid \boldsymbol{X}^O; \boldsymbol{\theta})$, does not decompose into a product over datapoints, i.e. $p(\boldsymbol{X}^M \mid \boldsymbol{X}^O; \boldsymbol{\theta}) \neq \prod_i p(\boldsymbol{X}_i^M \mid \boldsymbol{X}_i^O; \boldsymbol{\theta})$. Similarly, the loss optimized by NPTs does not decompose into a sum over individual datapoints. The direct dependence on other datapoints (and the ability to be trained on a single dataset) makes NPTs different from established deep learning architectures. We discuss this further, and relate it to IID-breaking in other approaches from the following chapters, in Chapter 7.

Limitations NPTs share scaling limitations with many non-parametric approaches (Rasmussen, 2003) and GNNs (Kipf and Welling, 2017). While we have seen success with random mini-batching (Section 3.2.6), future work might consider applying principled attention approximations, such as learning representative input points (Lee et al., 2019), kernelization (Katharopoulos et al., 2020; Choromanski et al., 2021), or other sparsity-inducing methods (Tay et al., 2020; Child et al., 2019; Beltagy et al., 2020) to improve the scalability of NPTs.

Future Work We believe that the unique predictive mechanism of NPTs makes them an interesting object of study for other tasks including semi-supervised learning, continual learning, multi-task learning, and domain adaptation. For example, when predicting under distribution shift, general relations between datapoints and attributes may remain valid and allow NPTs to accommodate such scenarios better.

3.6 Conclusions

This chapter has introduced Non-Parametric Transformers (NPTs), a novel data-efficient deep learning architecture that takes the entire dataset as input and uses self-attention to learn how to model complex relationships *between* datapoints. Concretely, NPTs have the additional flexibility to learn to predict by directly attending to other datapoints via self-attention layers. Notably, they learn how to do this – and implicitly even if it is beneficial to do so – end-to-end from the data at hand. Empirically, NPTs achieve highly competitive performance on tabular datasets. This is remarkable, as deep learning architectures have previously struggled to perform well on tabular datasets, in particular if the datasets are small. Additional experiments demonstrate NPTs' ability to solve complex reasoning tasks over datapoints. Further, we show that on real data, NPTs learn to rely on attention between datapoints for prediction.

In the next chapter, we switch gears and discuss an aspect of data-efficient machine learning that has previously often been overlooked: the cost of labeling datapoints for model *evaluation*.

4

Active Testing: Label-Efficient Model Evaluation

Contents

Authorship Statement

The work presented in this chapter has previously been published as Kossen*, Farquhar*, et al. (2021). Section 1.3 gives a detailed account of the contributions my collaborators have made to this work.

4.1 Introduction

Although unlabeled datapoints are often plentiful, labels can be expensive. For example, in scientific applications acquiring a single label can require expert researchers and weeks of lab time. However, some labels are more informative than others. In principle, this means that we can pick the most useful points to spend our budget wisely.

Such ideas have motivated extensive research into active learning, which we have previously discussed in Section 2.3. Active learning reduces the cost of *training* models by actively selecting which datapoints to acquire labels for. However,

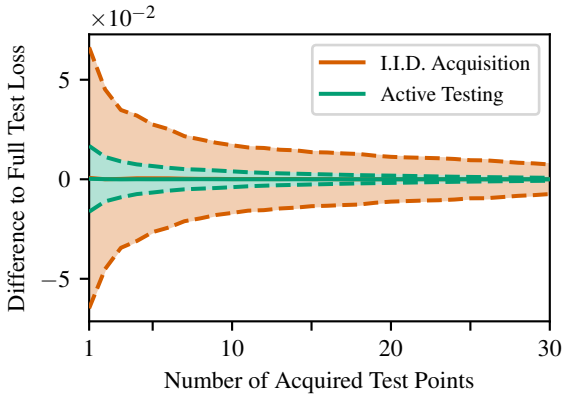


Figure 4.1: Active testing estimates the test loss much more precisely than uniform sampling for the same number of labeled test points. Active data selection during testing resolves a major barrier to sample-efficient machine learning, complementing prior work which has focused only on training. For details, see Section 4.5.1.

the cost of labeling *test* data has been largely ignored (Lowell et al., 2019). In artificial research settings, this is often not a problem: we can "cheat" by using enormous test datasets if the goal is to see how good some sample-efficient training approach is. But for practitioners this creates a huge issue: as we have discussed in Section 2.1.4, we must evaluate model performance in practice, for example to choose the best model or to develop trust in individual models. Whenever labels are expensive enough that we need to carefully pick training data, we cannot afford to be wasteful with test data either.

To address this, we introduce a framework for actively selecting test points for efficient labeling that we call *active testing*.[1] To enable active testing, we derive acquisition functions with which to select test points to maximize the accuracy of the resulting empirical risk estimate. We find that the principles that make these acquisition functions effective are quite different from their active learning counterparts. Given a fixed budget, we can then estimate quantities like the test loss much more accurately than naively labeling points at random. An example of this is given in Fig. 4.1.

We derive practical acquisition strategies for active testing by approximating an idealized approach. Each of these depends only on a predictive model for outputs, allowing for substantial customization of the framework to particular

[1] Code for active testing is available at github.com/jlko/active-testing.

problems and computational budgets. For example, we demonstrate both fast, simple, and surprisingly effective methods that rely only on the original model, as well as much more powerful techniques which use a 'surrogate' model that captures information from already acquired test labels and accounts for errors and potential overconfidence in the original model.

A difficulty in active testing is that choosing test data using information from the training data or the model being evaluated creates a sample-selection bias (MacKay, 1992c; Dasgupta and Hsu, 2008). For example, acquiring points where the model is least certain (Houlsby et al., 2011) will likely overestimate the test loss: the least certain points will tend to be harder than average. Moreover, the effect will be stronger for overconfident models, undermining our ability to select models or optimize hyperparameters. We show how to remove this bias using the weighting scheme introduced by Farquhar et al. (2021), which we have previously discussed briefly in Section 2.3. The recency of this technique is perhaps why the extensive literature on active learning has neglected actively selecting test data; this bias is far more harmful for testing than it is for training.

Our approach is general and applies to practically any machine learning model and task, not just settings where active learning is used. We show active testing of standard neural networks, Bayesian neural networks, Gaussian processes, and random forests from toy regression data to image classification problems like CIFAR-100.

In summary, the main contributions of this chapter are: **(1)** We formalize active testing as a framework for sample-efficient model evaluation (Section 4.2). **(2)** We derive principled acquisition strategies for regression and classification (Section 4.3). **(3)** We empirically show that active testing yields label-efficient and unbiased model evaluations, significantly reducing the number of labels required for testing (Section 4.5).

4.2 Active Testing

In this section, we introduce the active testing framework. For now, we set aside the question of how to design the acquisition scheme itself, which we cover later in

Section 4.3. We start with a model which we wish to evaluate, $f : \mathcal{X} \to \mathcal{Y}$, with inputs $\boldsymbol{x} \in \mathcal{X}$. Note that f is *fixed as given*: we will evaluate it, and it will not change during evaluation. We make very few assumptions about the model. It could be parametric or non-parametric, stochastic or deterministic, and it could be applied to any supervised machine learning task.

In Section 2.1.4, we have discussed general strategies for model evaluation. Our goal with active testing is now to estimate some model evaluation statistic in a *sample-efficient* way. Depending on the setting, this could be a test accuracy, mean squared error, negative log-likelihood, or something else. For sake of generality, we can write the "test loss" for arbitrary loss functions $\mathcal{L}$ evaluated over a test set $\mathcal{D}_{\text{test}}$ of size N as

$$\hat{R} = \frac{1}{N} \sum\nolimits_{i_n \in \mathcal{D}_{\text{test}}} \mathcal{L}(f(\boldsymbol{x}_{i_n}), y_{i_n}) \,. \tag{4.1}$$

This test loss is what we would be able to calculate if we possessed labels for every point in the test set, and it is an unbiased estimate for the true risk $r = \mathbb{E}\left[\mathcal{L}(f(\boldsymbol{X}), Y)\right]$. However, for active testing, we cannot afford to label all the test data. Instead, we can label only a subset $\mathcal{O} \subseteq \mathcal{D}_{\text{test}}$. Although we could choose all elements of $\mathcal{O}$ in a single step, doing so is sub-optimal as information from previously acquired labels can be used to select future points more effectively. Thus, we will pre-emptively introduce an index m tracking the labeling order: at each step m, we acquire the label y_{i_m} for the point with index i_m and add this point to $\mathcal{O}$.

4.2.1 A Naive Baseline

Standard practice ignores the cost of labeling the test set – it does not actively pick the test points. For a labeling budget M, this is equivalent to uniformly sampling a subset of the test data and then calculating the subsample empirical risk,

$$\hat{R}_{\text{IID}} = \frac{1}{M} \sum\nolimits_{i_m \in \mathcal{O}} \mathcal{L}(f(\boldsymbol{x}_{i_m}), y_{i_m}). \tag{4.2}$$

Uniform sampling guarantees that the data are independently and identically distributed (IID, cf. Section 2.1.2) so that the estimate is unbiased, $\mathbb{E}[\hat{R}_{\text{IID}}] = \hat{R}$,

and converges to the empirical test risk, $\hat{R}_{\text{IID}} \to \hat{R}$ as $M \to N$. However, although this estimator is unbiased, its *variance* can be high in the typical setting of $M \ll N$. That is, on any given run, the test loss estimated according to this method might be very different from $\hat{R}$, even though they will be equal in expectation.

4.2.2 Actively Sampling Test Points

To improve on this naive baseline, we need to reduce the variance of the estimator. A key contribution of this chapter is that this can be done by *actively selecting* the most useful test points to label. Unfortunately, doing this naively will introduce unwanted and highly problematic bias into our estimates.

In the context of pool-based active learning, Farquhar et al. (2021) showed that biases from active selection can be corrected by using a stochastic acquisition process and formulating an importance sampling estimator. Namely, they introduce an *acquisition distribution* $q(i_m)$ that denotes the probability of selecting index i_m to be labeled. They then compute the Monte Carlo estimator $\hat{R}_{\text{LURE}}$ which, in our active testing setting, takes the form

$$\hat{R}_{\text{LURE}} = \frac{1}{M} \sum_{m=1}^{M} v_m \mathcal{L}\left(f(\boldsymbol{x}_{i_m}), y_{i_m}\right), \tag{4.3}$$

where M is the size of $\mathcal{O}$, N is the size of $\mathcal{D}_{\text{test}}$, and

$$v_m = 1 + \frac{N-M}{N-m}\left(\frac{1}{(N-m+1)q(i_m)} - 1\right). \tag{4.4}$$

Not only does $\hat{R}_{\text{LURE}}$ correct the bias of active sampling, if the proposal $q(i_m)$ is suitable it can also (drastically) reduce the variance of the resulting risk estimates compared to both $\hat{R}_{\text{IID}}$ as well as naively applying active sampling without bias correction. This is because $\hat{R}_{\text{LURE}}$ is based on importance sampling: a technique designed precisely for reducing variance through appropriate proposals (Kahn and Marshall, 1953; Kahn, 1955; Owen, 2013).

Importantly, there are no restrictions on how $q(i_m)$ can depend on the data, and in our context $q(i_m)$ is actually shorthand for $q(i_m; i_{1:m-1}, \mathcal{D}_{\text{test}}, \mathcal{D}_{\text{train}})$. This means

that we will be able use proposals that depend on previously acquired test data, as well as the training data and the trained model, as we explain in the next section.

While we refer to Farquhar et al. (2021) for the derivation of the estimator and proof of its unbiasedness and variance, we here offer some intuitions as to its general form. For standard importance sampling *with* replacement from a pool of N samples, the acquisition weights are given by $p(\boldsymbol{x})/q(\boldsymbol{x}) = 1/(Nq(\boldsymbol{x}))$. For $\hat{R}_{\text{LURE}}$, which assumes importance sampling *without* replacement, we recover these weights for $M = m = 1$. For $m > 1$, the support of q shrinks with each acquisition, and so a naive importance sampling estimate is no longer appropriate. The extra terms in Eq. (4.4) correct for this and guarantee that $\hat{R}_{\text{LURE}}$ is unbiased when sampling from pools without replacement. They also ensure that $\hat{R}_{\text{LURE}} = \hat{R}_{\text{IID}}$ for $M = N$ and that weights are equal for uniform proposals, $q(i_m) = 1/(N - m + 1)$.

4.3 Acquisition Functions for Active Testing

In the last section, we showed how to construct an unbiased estimator of the test risk using actively sampled test data. This is exactly the quantity that the practitioner cares about for evaluating a model. For an estimator to be sample-efficient, its variance should be as small as possible for any given number of labels, M. We now use this principle to derive acquisition proposal distributions (i.e. acquisition functions) for active testing by constructing an idealized proposal and then showing how it can be approximated.

4.3.1 General Framework

As shown by Farquhar et al. (2021), the optimal oracle proposal for $\hat{R}_{\text{LURE}}$ is to sample in proportion to the true loss of each data point, resulting in a single-sample zero-variance estimate of the risk. In practice, we cannot know the true loss before we have access to the actual label. In particular, the true distribution of outputs for a given input is typically noisy and we can never know this noise without evaluating

the label. In the context of deriving an unbiased Monte Carlo estimator, the best we can ever hope to achieve is to sample from the expected loss over the true $Y \mid \boldsymbol{x}_{i_m}$,[2]

$$q^*(i_m) \propto \mathbb{E}_{Y \sim p(\cdot \mid \boldsymbol{x}_{i_m})} \left[\mathcal{L}(f(\boldsymbol{x}_{i_m}), Y) \right]. \tag{4.5}$$

Note that as i_m can only take on a finite set of values, the required normalization can be performed straightforwardly.

Of course, $q^*(i_m)$ remains intractable because we do not know the true distribution for $Y \mid \boldsymbol{x}_{i_m}$. We need to approximate it for unlabeled $\boldsymbol{x}_{i_m}$ in a way that captures regions where $f(\boldsymbol{x})$ is a poor predictive model as these will contribute the most to the loss. This can be hard as $f(\boldsymbol{x})$ itself has already been designed to approximate $Y \mid \boldsymbol{x}$.

Thankfully, we have the following tools at our disposal to deal with this: (a) We can incorporate *uncertainty* to identify regions with a lack of available information (e.g. regions far from any of the training data); (b) We can introduce *diversity* in our predictions compared to $f(\boldsymbol{x})$ (thereby ensuring that mistakes we make are as distinct as possible to those of $f(\boldsymbol{x})$); and (c) as we label new points in the test set, we can obtain *more accurate* predictions than $f(\boldsymbol{x})$ by incorporating these additional points. These essential strategies will help us identify regions where $f(\boldsymbol{x})$ provides a poor fit. We give examples of how we incorporate them in practice in Section 4.3.4.

We now introduce a general framework for approximating $q^*(i_m)$ that allows us to use these mechanisms as best as possible. The starting point for this is to consider the concept of a *surrogate* for $Y \mid \boldsymbol{X} = \boldsymbol{x}$, where we introduce some potentially infinite set of parameters $\boldsymbol{\theta}$, a corresponding model $\pi(\boldsymbol{\theta})\pi(y \mid \boldsymbol{x}, \boldsymbol{\theta})$, and then approximate the true $p(y \mid \boldsymbol{x})$ using the marginal distribution $\pi(y \mid \boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\Theta} \sim \pi(\cdot)} \left[\pi(y \mid \boldsymbol{x}, \boldsymbol{\Theta}) \right]$ of the surrogate. We can now approximate $q^*(i_m)$ as

$$q(i_m) \propto \mathbb{E}_{\boldsymbol{\Theta} \sim \pi(\cdot)} \left[\mathbb{E}_{Y \sim \pi(\cdot \mid \boldsymbol{x}_{i_m}, \boldsymbol{\Theta})} \left[\mathcal{L}(f(\boldsymbol{x}_{i_m}), Y) \right] \right]. \tag{4.6}$$

With $\boldsymbol{\theta}$ we represent our subjective, i.e. Bayesian, uncertainty over the outcomes in a principled way, cf. our discussions in Chapter 2. However, our derivations will lead to acquisition strategies which are also compatible with non-Bayesian models.

[2] For estimators other than $\hat{R}_{\text{LURE}}$, e.g. ones based on direct regression of the loss, this may no longer be true, cf. Chapter 5.

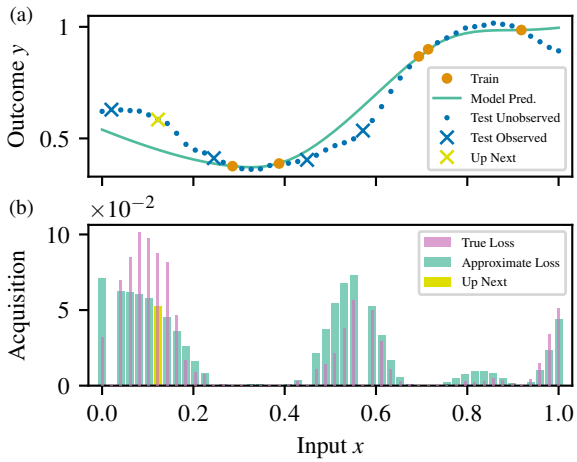


Figure 4.2: Illustration of a single active testing step. (a) The model has been trained on five points, and we currently have observed four test points. (b) We assign acquisition probabilities using the estimated loss of potential test points. Because we do not have access to the true labels, these estimates are different from the true loss. Our next acquisition is then *sampled* from this distribution.

4.3.2 Illustrative Example

Figure 4.2 shows how active testing chooses the next test point among all available test data. The model, here a Gaussian process (Rasmussen (2003), Section 2.1.3), has been trained using the training data and we have already acquired some test points (crosses). Fig. 4.2 (b) shows the true loss known only to an oracle. Our approximate expected loss is a good proxy in some parts of the input space and worse elsewhere. The next point is selected by sampling proportionately to the approximate expected loss. In this example, the surrogate is a Gaussian process that is retrained whenever new labels are observed. The closer the approximate expected loss is to the true loss, the lower the variance of the estimator $\hat{R}_{\mathrm{LURE}}$ will be; the estimator will always be unbiased.

4.3.3 Deriving Acquisition Functions

We now give principled derivations leading to acquisition functions for a variety of widely-used loss functions.

Regression Substituting the *squared error loss*, $\mathcal{L}(f(\boldsymbol{x}), y) = (f(\boldsymbol{x}) - y)^2$, into Eq. (4.6) yields

$$q(i_m) \propto \mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x}_{i_m})} \left[(f(\boldsymbol{x}_{i_m}) - Y)^2 \right], \tag{4.7}$$

and if we apply a bias-variance decomposition this becomes

$$q(i_m) \propto \underbrace{(f(\boldsymbol{x}_{i_m}) - \mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x}_{i_m})}[Y])^2}_{①} + \underbrace{\mathbb{V}_{Y \sim \pi(\cdot|\boldsymbol{x}_{i_m})}[Y]}_{②}. \tag{4.8}$$

Here, ① is the squared difference between our model prediction $f(\boldsymbol{x}_{i_m})$ and the mean prediction of the surrogate: it measures how wrong the surrogate believes the prediction to be. ② is the predictive variance: the uncertainty that the surrogate has about Y at $\boldsymbol{x}_{i_m}$.

Both ① and ② are readily accessible in models such as Gaussian processes or Bayesian neural networks. However, we actually do not need an explicit $\pi(\boldsymbol{\theta})$ to acquire with Eq. (4.8): we only need to provide approximations for the mean prediction $\mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x}_{i_m})}[Y]$ and predictive variance $\mathbb{V}_{Y \sim \pi(Y|\boldsymbol{x}_{i_m})}[Y]$. For example, these exist for "deep ensembles" (Lakshminarayanan et al., 2017) that compute mean and variance predictions from a set of standard neural networks.

A critical subtlety to appreciate is that ② incorporates both aleatoric *and* epistemic uncertainty (Kendall and Gal, 2017). It is not our estimate for the level of noise in $Y \mid \boldsymbol{x}_{i_m}$ but the variance of our subjective beliefs for what the value of Y *could* be at $\boldsymbol{x}_{i_m}$. This is perhaps easiest to see by noting that

$$\mathbb{V}_{Y \sim \pi(\cdot|\boldsymbol{x}_{i_m})}[Y] = \mathbb{V}_{\boldsymbol{\Theta} \sim \pi(\cdot)} \left[\mathbb{E}_{\pi(Y|\boldsymbol{x}_{i_m}, \boldsymbol{\Theta})}[Y] \right] + \mathbb{E}_{\boldsymbol{\Theta} \sim \pi(\cdot)} \left[\mathbb{V}_{\pi(Y|\boldsymbol{x}_{i_m}, \boldsymbol{\Theta})}[Y] \right], \tag{4.9}$$

where the first term is the variance in our mean prediction and represents our epistemic uncertainty and the latter is our mean prediction of the "aleatoric variance", which represents our estimate of label noise. This is why our construction using $\boldsymbol{\theta}$ is crucial: it stresses that Eq. (4.8) should also take epistemic uncertainty into account.

For regression models with Gaussian outputs $\mathcal{N}(f(\boldsymbol{x}), \sigma^2)$, the *negative log-likelihood loss* function and the squared error are related by affine transformation – and following Eq. (4.6) so are the acquisition functions.

Classification For classification, predictions $f(\boldsymbol{x})$ generally take the form of conditional probabilities over outcomes $y \in \{1, \ldots, C\}$. First, we study *cross-entropy*,

$$\mathcal{L}(f(\boldsymbol{x}), y) = -\log f(\boldsymbol{x})_y. \tag{4.10}$$

We again introduce a surrogate, and, using Eq. (4.6), obtain

$$q(i_m) \propto \mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x}_{i_m})} \left[-\log f(\boldsymbol{x}_{i_m})_Y \right]. \tag{4.11}$$

Now expanding the expectation over Y yields

$$q(i_m) \propto -\sum\nolimits_y \pi(y \mid \boldsymbol{x}_{i_m}) \log f(\boldsymbol{x}_{i_m})_y, \tag{4.12}$$

which is the cross-entropy between the marginal predictive distribution of our surrogate, $\pi(y \mid \boldsymbol{x}_{i_m})$, and our model.

We can also derive acquisition strategies based on *accuracy*. Namely, writing one minus accuracy to obtain a loss,

$$\mathcal{L}(f(\boldsymbol{x}), y) = 1 - \mathbb{1}[y = \arg\max_{y'} f(\boldsymbol{x})_{y'}], \tag{4.13}$$

and substituting into Eq. (4.6) yields

$$q(i_m) \propto 1 - \pi(y = y^*(\boldsymbol{x}_{i_m}) \mid \boldsymbol{x}_{i_m}), \tag{4.14}$$

where $y^*(\boldsymbol{x}_{i_m}) = \arg\max_{y'} f(\boldsymbol{x}_{i_m})_{y'}$.

4.3.4 Tactics for Obtaining Good Surrogates

In Section 4.3.1 we introduced three ways for the surrogate to assist in finding high-loss regions of $f(\boldsymbol{x})$: we want it to (a) account for uncertainty over the outcomes, (b) make predictions that are diverse to $f(\boldsymbol{x})$, and (c) incorporate information from all available data. Motivated by this, we apply the following tactics to obtain good surrogates:

Uncertainty We should use surrogates that incorporate both epistemic and aleatoric uncertainty effectively, and further ensure that these are well-calibrated. Capturing epistemic uncertainty is essential to predicting regions of high loss, while aleatoric uncertainty still contributes and cannot be ignored, particularly if heteroscedastic. A variety of different approaches can be effective in this regard and thus provide successful surrogates. For example, Bayesian neural networks, deep ensembles, and Gaussian processes.

Fidelity In real-world settings, f may be constrained to be memory-efficient, fast, or interpretable. If labels are expensive enough, we can relax these constraints at test time and construct a more capable surrogate. In fact, we practically find that using an ensemble of models like f is a robust way of achieving sample-efficiency.

Diversity By choosing the surrogate from a different model family or adjusting its hyperparameters, we can decorrelate the errors of the surrogate and f, resulting in better exploration. For example, we find that random forests (Breiman, 2001) can help evaluate neural networks.

Extra data If our computational constraints are not critical, we should retrain the surrogate on $\mathcal{O} \cup \mathcal{D}_{\text{train}}$ after each step. The exposure to additional data will make the surrogate a better approximation of the true outcomes. With retraining, the acquisition proposal changes between iterations, meaning the data are no longer sampled IID. Active testing then forms an *adaptive* importance sampling estimate (Karamchandani et al., 1989; Oh and Berger, 1992; Bugallo et al., 2017). On the other hand, when the acquisition proposal is not adaptive, the data are in fact collected in IID fashion – but from $q(i_m)p(y_{i_m} \mid \boldsymbol{x}_{i_m})$ instead of the underlying data-generating distribution. While we generally advise against such fixed proposals, we will later, in Section 4.5.3, see that this basic approach can sometimes succeed, and Yilmaz et al. (2021) similarly find fixed proposals can be effective. For simplicity, we always refer to our LURE-based estimation approach as *active testing* and to naive Monte Carlo subsampling according to Eq. (4.2) as *IID Acquisition*.

Algorithm 1: Active Testing

Input: Model f trained on data $\mathcal{D}_{\text{train}}$

Train surrogate π & choose acquisition proposal form q

for $m = 1$ *to* M **do**

  $i_m \sim q(i_m; \pi)$, observe y_{i_m}, add to $\mathcal{O}$

  Calculate $\mathcal{L}(f(\boldsymbol{x}_{i_m}), y_{i_m})$ and v_m // Eq. (4.4)

  Update π, e.g. retraining on $\mathcal{D}_{\text{train}} \cup \mathcal{O}$

end

return $\hat{R}_{\text{LURE}}$ // Eq. (4.3)

Thompson-Ensemble Retraining the surrogate can also create diversity in predictions due to stochasticity in the training process, the addition of new data, or even deliberate randomization. In fact, we can view retraining the surrogate at regular intervals as implicitly defining an ensemble of surrogates, with the surrogate used at any given iteration forming a Thompson-sample (Thompson, 1933) from this ensemble. This will generally be more powerful and more diverse than a single surrogate, providing further motivation for retraining and potentially even deliberate variations in surrogates/hyperparameters between iterations.

In Section 4.5 we empirically assess the relative importance of these considerations, which depends heavily on the situation. For example, the benefit of retraining using the labels acquired at test-time is especially large in very low-data settings, while the benefit of ensembling can be large even when there is more data available. Putting everything together, Algorithm 1 provides a summary of our general framework.

If compute is at a premium for acquisitions, a simple alternative heuristic is to use our original model for the surrogate. This avoids learning a new predictive model, but it suffers because now the surrogate can never disagree with f. Instead, we have to rely entirely on uncertainties for approximating Eq. (4.6): for regression, ① in Eq. (4.8) is zero, and for classification, Eq. (4.12) reduces to the predictive entropy. In general, we do not recommend this strategy, *unless* computational constraints are substantial relative to labeling costs *and* there is reason to believe that the epistemic and aleatoric uncertainties from f represent the true loss well. If

Why are acquisition strategies different for active learning and active testing?

Researchers have already investigated acquisition functions for active learning, and it would be helpful if we could just apply these here. However, active testing is a different problem conceptually because we are not trying to use the data to fit a model.

First, popular approaches for active learning avoid areas with high aleatoric uncertainty while seeking out high epistemic uncertainty. This motivates acquisition functions like BALD (Houlsby et al., 2011) or BatchBALD (Kirsch et al., 2019). For active testing, however, areas of high aleatoric uncertainty can be critical to the estimate.

Second, as Imberg et al. (2020) point out, the optimal acquisition scheme for active learning will minimize the expected generalization error at the end of training. They show how this motivates additional terms beyond what one would get from minimizing the variance of the loss estimator.

Third, as Farquhar et al. (2021) show, a biased loss estimator can be helpful during training because it often partially cancels the natural bias of the training loss. This is no longer true at test-time, where we want to minimize bias as much as possible.

the latter is true, this simplistic approach can perform surprisingly well, although it is always outperformed by more complex strategies. In particular, training a single fixed surrogate that is distinct from f will still typically provide noticeable benefits.

4.4 Related Work

The efficient use of labels is a major aim in machine learning, and we refer to Section 2.3 for a general overview of strategies for label-efficient machine learning.

What most of this work neglects is the wasteful acquisition of data for *testing*. Lowell et al. (2019) acknowledge this and describe it as a major barrier to the adoption of active learning methods in practice. The potential for "active testing" was raised by Nguyen et al. (2018), but they focused on the special case of noisily annotated labels that must be vetted and did not acknowledge the substantial bias that their method introduces. Farquhar et al. (2021) introduce the variance-reducing unbiased estimator for active sampling which we apply. However, their focus is mostly on correcting the bias of active *learning* (Bach, 2007; Sugiyama,

2006; Beygelzimer et al., 2009; Ganti and Gray, 2012) and they do not consider appropriate acquisition strategies for active testing. Note that their theoretical results about the properties of $\hat{R}_{\text{LURE}}$ carry over to active testing.

Other methods like Bayesian Quadrature (Rasmussen and Ghahramani, 2003; Osborne, 2010) and kernel herding (Chen et al., 2012) can also sometimes employ active selection of points. Of particular note, Osborne et al. (2012) and Chai et al. (2019) study active learning of model evidence in the context of Bayesian Quadrature.

Bennett and Carvalho (2010), Katariya et al. (2012), Kumar and Raj (2018), and Ji et al. (2021) explore the efficient evaluation of classifiers based on stratification, rather than active selection of individual labels. Namely, they divide the test pool into strata according to simple metrics such as classifier confidence. Test data are then acquired by first sampling a stratum and then selecting data *uniformly* within. Sample-efficiency for these approaches could be improved by performing active testing within the strata. Sawade et al. (2010) similarly explore active risk estimation through importance sampling, but rely on sampling with replacement which is suboptimal in pool-based settings (see Appendix B.4). Yilmaz et al. (2021) have recently proposed active testing based on Poisson sampling. Both (Yilmaz et al., 2021) and (Sawade et al., 2010) outperform simple baselines; however, they use *non-adaptive* acquisitions that cannot adjust to test data.

4.5 Experiments

We now assess the empirical performance of active testing and investigate the relative merits of different strategies. Similar to active learning, we assume a setting where sample acquisition is expensive, and therefore, per-sample efficiency is critical.

We note a small but important practicality: we ensure all points have a minimum proposal probability regardless of the acquisition function value, to ensure that the weights are bounded even if q is badly calibrated (cf. Appendix B.2.1).

4.5.1 Synthetic Data

We first show that active testing on synthetic datasets offers sample-efficient model evaluations. We actively evaluate a Gaussian process (Rasmussen (2003), Section 2.1.3) and a linear model for regression, and a random forest (Breiman, 2001) for classification. For regression, we estimate the squared error and acquire test labels via Eq. (4.8); for classification, we estimate the cross-entropy loss and acquire with Eq. (4.12). We use Gaussian process and random forest surrogates that are retrained on all observed data after each acquisition.

Figure 4.3 shows how the difference between our test loss estimation and the truth (known only to an oracle) is much smaller than the naive $\hat{R}_{\text{IID}}$: active testing allows us to precisely estimate the empirical test loss using far fewer samples. For example, after acquiring labels for only 5 test points in (a), the standard deviation of active testing is already as low as it is for naive IID acquisition at step 40, nearly the entire test set. Further, we can see that the estimates of $\hat{R}_{\text{LURE}}$ are indeed unbiased. Appendix B.1.1 gives experiments on additional synthetic datasets.

Here we have actually acquired the full test set. This lets us show that both $\hat{R}_{\text{LURE}}$ and $\hat{R}_{\text{IID}}$ converge to the empirical test loss on the entire test set. However, typically we cannot do this which makes the difference in variance between $\hat{R}_{\text{IID}}$ and $\hat{R}_{\text{LURE}}$ at lower acquisition numbers crucial.

4.5.2 Surrogate Choice Case Study: Image Classification

We now investigate the impact of the different surrogate choices. For this, we move to more complex image classification tasks and additionally restrict the number of training points to only 250. This makes it harder to predict the true loss. Therefore, the strategies discussed in Section 4.3.4 are especially important to maximize sample-efficiency.

We evaluate two model types for this examination. First, a Radial Bayesian Neural Network (Radial BNN) (Farquhar et al., 2020) on the MNIST dataset (LeCun et al., 1998) in Fig. 4.4 (a). Radial BNNs are a recent approach for variational

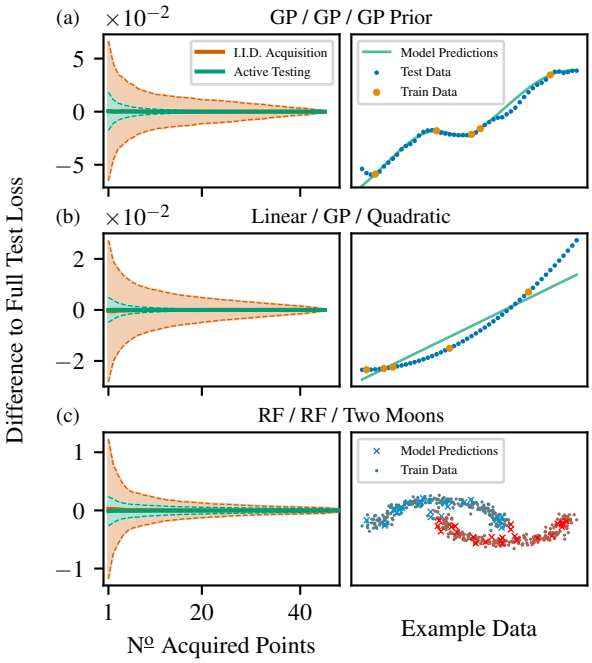


Figure 4.3: Active testing yields unbiased estimates of the test loss with significantly reduced variance. Each row shows a different combination of *model/surrogate/data*. GP is short for Gaussian process, RF for random forest. The first column displays the mean difference of the estimators to the true loss on the full test set (known only to an oracle). We retrain surrogates after each acquisition on all observed data. Shading indicates standard deviation over 5000 (a-b) / 2500 (c) runs, where data is randomized between runs. The second column shows example data with model predictions, and the points used for training and testing (a-b).

inference in BNNs (Blundell et al., 2015a) and we use them because of their well-calibrated uncertainties, cf. Section 2.3. We also evaluate a ResNet-18 (He et al., 2016) trained on Fashion-MNIST (Xiao et al., 2017) in Fig. 4.4 (b) to investigate active testing with convolutional neural network (CNN) architectures.[3] In these figures, we show the median squared error of the different surrogate strategies on a logarithmic scale to highlight differences between the approaches. While not shown, note that all approaches do still obtain unbiased estimates. We again use Eq. (4.12) to estimate the cross-entropy loss of the models.

Predictive Entropy We first consider the most naive of the approaches mentioned in Section 4.3.4: using the unchanged original model as the surrogate, which leads to acquisitions based on model predictive entropy. For the Radial BNN, this approach

[3] We refer back to Section 2.2 for a general overview of deep learning models, including BNNs and CNNs.

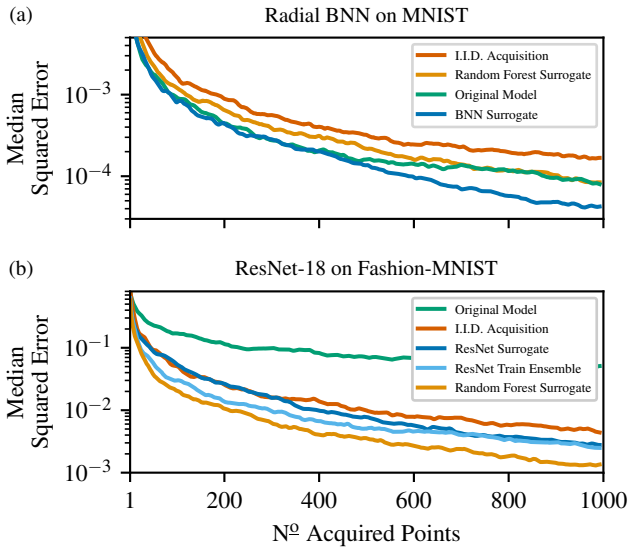


Figure 4.4: Median squared errors for (a) Radial BNN on **MNIST** and (b) ResNet-18 on **Fashion-MNIST** in a small-data setting. Original Model samples proportional to predictive entropy, X Surrogate iteratively retrains a surrogate on all observed data, and ResNet Train Ensemble is a deep ensemble trained on $\mathcal{D}_{\text{train}}$ once. Lower is better; medians are over 1085 runs for (a), 872 for (b).

already yields improvements over naive IID acquisition in Fig. 4.4 (a). The same can not be said for the ResNet in Fig. 4.4 (b), for which predictive entropy actually performs worse than naive IID acquisition. Presumably, this is the case because the standard neural network struggles to model epistemic uncertainty.

Retraining Next, we progress to more complex surrogates. BNN surrogate is a surrogate with identical setup as the original model that is retrained on the total observed data, $\mathcal{D}_{\text{train}} \cup \mathcal{O}$, 12 times with increasingly large gaps. This leads to improved performance over the naive model predictive entropy, especially as more data is acquired. Similarly, the ResNet surrogate shows much-improved performance over predictive entropy when regularly retrained, now outperforming IID acquisition.

Different Model As discussed in Section 4.3.4, it may be beneficial to choose the surrogate from a different model family to promote diversity in its predictions. We use a random forest as a surrogate for both Fig. 4.4 (a) and (b). For the Radial BNN on MNIST, the random forest, while better than IID acquisition, does not improve over the model predictive entropy. However, for the ResNet on Fashion-MNIST,

we find that the random forest surrogate outperforms everything, despite being a cheaper surrogate. This demonstrates that for a surrogate to be successful, it does not necessarily need to be more accurate – although the difference in accuracy is small with so few data. Instead, the surrogate can be also be successful by being *different* from the original model, i.e. having structural biases that lead to it making different predictions and therefore discovering mistakes of the original model, with any new mistakes made less important. Further, if compute is limited, the random forest is attractive because retraining it is much faster.

Ensembling Diversity Section 4.3.4 discussed two ways retraining may help: new data improves the surrogate's predictive model and repeated training promotes diversity through an implicit ensemble. In Fig. 4.4 (b), we introduce the ResNet train ensemble – a deep ensemble of ResNets trained once on $\mathcal{D}_{\text{train}}$. This surrogate allows us to isolate the effect of predictive diversity since it is not exposed to any test data through retraining. We output mean predictions of the ensemble, and find that the deep ensemble can, a little unexpectedly, outperform the ResNet surrogate without accessing the extra data. This is likely because of better calibrated uncertainties and the increased model capacity.

In summary, we have shown that active testing reduces the number of labeled examples required to get a reliable estimate of the test loss for a Radial BNN on MNIST and ResNet-18 on FashionMNIST in a challenging setting, if appropriate surrogates are chosen.

4.5.3 Large-Scale Image Classification

We now additionally apply active testing to a Resnet-18 trained on CIFAR-10 (Krizhevsky, Hinton, et al., 2009) and a WideResNet (Zagoruyko and Komodakis, 2016) trained on CIFAR-100. As the complexity of the datasets increases, it becomes harder to estimate the loss, and hence, it is crucial to show that active testing scales to these scenarios. We use conventional training set sizes of 50 000 data points.

In the previous section, we have seen surrogates based on deep ensembles perform well, even if they are only trained once and not exposed to any acquired test labels. For the following experiments, we therefore use these ensembles as surrogates. This is even more justified in the common case where there is much more training data than test data; the extra information in the test labels will typically help less.

In Fig. 4.5, we further visualize how ensembles increase the quality of the approximated loss in this setting. The original model (b) makes overconfident false predictions with high losses which are rarely detected (box). But the ensemble avoids the majority of these mistakes (c, box) which contribute most to the weighted loss estimator Eq. (4.3).

In all cases, the active testing estimator has lower median squared error than the baseline, see Fig. 4.6 (a) – again note the log-scale. We further show in Fig. 4.6 (b) that using active testing is much more sample-efficient than IID sampling by calculating the "relative labeling cost": the proportion of actively chosen points needed to get the same performance as naive uniform sampling. E.g., a cost of 0.25 means we need only 1/4 of actively chosen labels to get equivalently precise test loss. Thus, for the less complex datasets, we see efficiency gains are in the region of a factor of four, while for CIFAR-100 they are closer to a factor of two. We also show that there are similar gains in sample-efficiency when estimating accuracy, see CIFAR-10 Accuracy in Fig. 4.6 (b).

4.5.4 Diversity and Fidelity in Ensemble Surrogates

We now perform an ablation study of the relative effects of surrogate fidelity and diversity on active testing performance. For this, we evaluate a ResNet-18 trained on CIFAR-10 using different ResNet ensembles as surrogates. Starting with a base surrogate of a single ResNet-18, we increase the size of the ensemble (mainly increasing diversity) as well as the capacity of the layers (increasing fidelity). Given the success of the "Train Ensemble" in Section 4.5.2, we train surrogates only once on $\mathcal{D}_{\text{train}}$, rather than retraining as data is acquired.

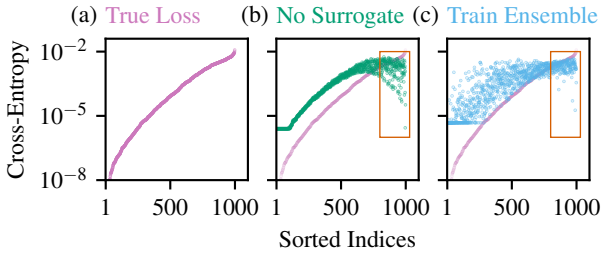


Figure 4.5: Predictive entropy underestimates the true loss of some points by orders of magnitude. Diverse predictions from the ensemble of surrogates help for these crucial high-confidence mistakes, even though they are noisier for low-loss points, improving sample-efficiency overall. (a) We sort values of the true losses and use the index order to plot (b) the approximate losses for predictive entropy and (c) an ensemble of surrogates, ideally seeing few small approximated losses on the right. Shown is a ResNet-18 on CIFAR-100; note the log-scale on y and the use of clipping to avoid overly small acquisition probabilities.

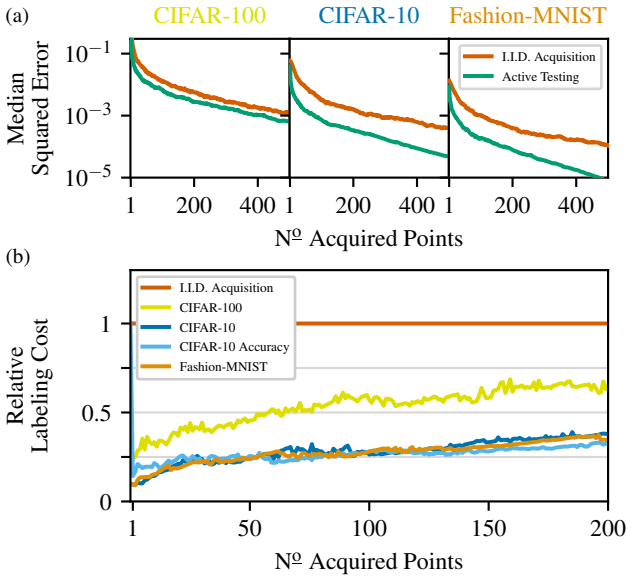


Figure 4.6: Active testing of a WideResNet on **CIFAR-100** and ResNet-18 on **CIFAR-10** and **Fashion-MNIST**. (a) Convergences of errors for active testing/IID acquisition. (b) Relative effective labeling cost. Active testing consistently improves the sample-efficiency. Lower is better; medians over 1000 random test sets.

As Fig. 4.7 shows, both fidelity and diversity contribute to active testing performance: the best performance is obtained for the most diverse and complex surrogate, justifying our claims in Section 4.3.4. We see that increasing fidelity and diversity both individually help performance, with the effect of the latter seeming to be slightly more pronounced (e.g. an ensemble of 5 Resnet-18s outperforms a single ResNet-50).

4.5.5 Optimal Proposals and Unbiasedness

Fig. 4.8 (a) confirms our theoretical assumptions by showing that sampling proportional to the true loss, i.e. cheating by asking an oracle for the true outcomes beforehand, does indeed yield exact, single-sample estimates of the loss if combined with $\hat{R}_{\text{LURE}}$. Further, it confirms the need for a bias-correcting estimator such as $\hat{R}_{\text{LURE}}$: without it, the risk estimates are biased and clearly overestimate model error.

4.5.6 Active Testing vs. Active Learning

As mentioned in Section 4.3.4, we expect there to be differences in acquisition function requirements for active learning and active testing. For example, mutual information is a popular acquisition function in active learning (Houlsby et al., 2011), cf. Section 2.3, but our derivations for classification lead to acquisition strategies based on predictive entropy. Can mutual information also be used for active testing? In Fig. 4.8 (b) we see that even the simple approach of using the original model as a surrogate and a predictive entropy acquisition outperforms mutual information. Acquiring with mutual information helps active *learning* because it focuses on uncertainty that can be reduced by more information rather than irreducible noise. While this focus helps learning, it is unhelpful for *evaluation* where all uncertainty is relevant, cf. Eq. (4.9). This is just one way active testing needs special examination and cannot just re-use results from active learning.

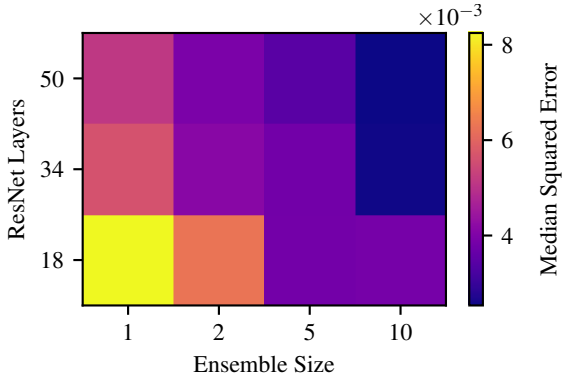


Figure 4.7: Both diversity and fidelity of the surrogate contribute to sample-efficient active testing. However, the effect of increasing diversity seems larger than that of increased fidelity. We vary the layers (fidelity) and ensemble size (diversity) of the surrogate for active evaluation of a ResNet-18 trained on CIFAR-10. Experiments are repeated for 1000 randomly drawn test sets and we report average values over acquisition steps 100–200.

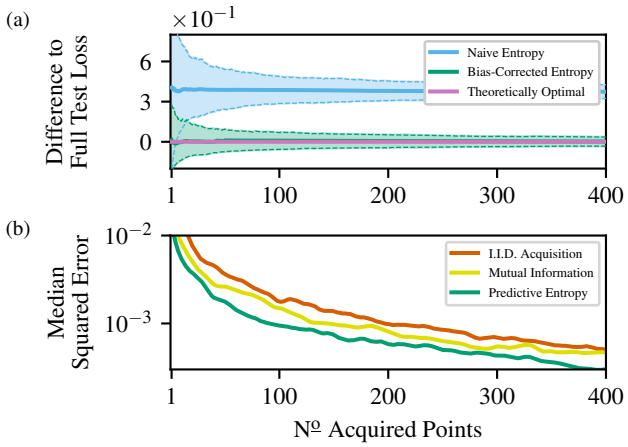


Figure 4.8: (a) Naively acquiring proportional to the predictive entropy and using the unweighted estimator $\hat{R}_{\text{IID}}$ leads to biased estimates with high variance compared to active testing with $\hat{R}_{\text{LURE}}$. Sampling from the unknown true loss distribution would yield unbiased, zero-variance estimates when used with $\hat{R}_{\text{LURE}}$. While this is in practice impossible, the result validates a main theoretical assumption. (b) Mutual information, popular in active learning, underperforms for active testing, even compared to the simple predictive entropy approach. This is because it does not target expected loss. Shown for 692 runs of a Radial BNN on Fashion-MNIST.

4.6 Discussion

Empirically, we find that deep ensemble surrogates appear to robustly achieve sample-efficient active testing when using our acquisition strategies. Increases in surrogate fidelity further seem to benefit sample-efficiency.

Active testing generally assumes that acquisitions of labels for samples are expensive, hence we recommend retraining the surrogate whenever new data becomes available. However, if the cost of this is noticeable relative to that of labeling, our results indicate that not retraining the surrogates is an option, especially when the number of acquired test labels is small compared to the training data.

In general, we do not recommend the naive strategy that relies entirely on the original model and does not introduce a dedicated surrogate model. As Section 4.5.2 has shown, this method can fail to achieve sample-efficient active testing if the original model does not have trustworthy uncertainties. This strategy should remain a last resort and used only when there is significant reason to trust the original model's uncertainties; we find the diversity provided by a surrogate is critical, even if that surrogate is itself simple.

4.7 Conclusions

We have introduced the concept of active testing and given principled derivations for acquisition functions suitable for model evaluation. Active testing uses ideas from adaptive importance sampling to yield more precise estimates of test loss and accuracy using fewer data labels. While our work provides an exciting starting point for active testing, we believe that the underlying idea of sample-efficient evaluation leaves significant scope for further development and alternative approaches. In the next chapter, we propose an alternative to importance sampling–based active testing that directly predicts the loss of test points.

5
Active Surrogate Estimators

Contents

Authorship Statement

The work presented in this chapter has previously been published as Kossen et al. (2022). Section 1.3 gives a detailed account of the contributions my collaborators have made to this work.

5.1 Introduction

In the previous chapter, we have proposed a method for label-efficient model evaluation based on adaptive importance sampling (AIS), improving on previous (non-adaptive) importance sampling (IS) strategies (Sawade et al., 2010). However, even AIS-based approaches to label-efficient model evaluation have a number of weaknesses, that suggest further improvements should be possible.

Firstly, the AIS weights assigned to samples depend on the acquisition proposal used *at that iteration*. Thus, early samples are often inappropriately weighted, because we are yet to learn an effective proposal. This can significantly increase the variance of the resulting estimator. Secondly, MC estimates can be inefficient at dealing with label noise: constraining the estimate to be a weighted sum of observed losses is limiting when the losses themselves are noisy. Finally, IS methods can be

restrictive on the acquisition strategies used to choose points to label: acquisition must be stochastic with known probabilities.

To address these limitations, we propose Active Surrogate Estimators (ASEs), a new approach to label-efficient model evaluation that is based on interpolation rather than MC estimation.[1] Specifically, ASEs actively learn a surrogate model to directly *predict* losses at all observed *and unobserved* points in the pool. Here, the surrogate can take the form of any appropriate machine learning model, and we introduce a general framework for learning effective surrogates. The interpolation-based nature of ASEs leads to a variety of advantages over MC estimates: ASEs better accommodate noisy label observations, adapt quicker to new information, and provide complete flexibility in how new datapoints are acquired, allowing for careful customizations that are not appropriate in the AIS framework of Chapter 4. To exploit this, we propose a novel acquisition strategy, XWED, that tailors the acquisition towards points that will be most beneficial to the final estimation task.

To summarize, the main contributions of this chapter are: **(1)** We introduce Active Surrogate Estimators; a new method for label-efficient model evaluation based on actively learning a surrogate model (Section 5.2). **(2)** We introduce XWED, a novel acquisition strategy that targets a weighted disagreement that is highly relevant to our final surrogate-based prediction (Section 5.3). **(3)** We provide theoretical insights into ASEs' errors (Section 5.4). **(4)** We apply ASEs to the evaluation of deep neural networks trained for image classification, where we consistently outperform the current state-of-the-art (Section 5.6).

5.2 Active Surrogate Estimators

Following the same setup as in Section 4.2, we assume a pool of unlabeled samples from which we iteratively choose points to label. Unlike existing methods that rely on MC estimates, ASEs use an interpolation-based approach to estimate model risk: a surrogate model directly predicts losses for *all* points in the test pool.

[1] Code for Active Surrogate Estimators is available at github.com/jlko/active-surrogate-estimators.

ASEs are further based around an adaptive and iterative procedure: we use an active learning approach to carefully select the labels used to train our surrogate. We first discuss the form that the ASE and surrogate take and then present our novel acquisition strategy in Section 5.3.

5.2.1 Deriving the ASE

The fundamental idea of ASEs is that of interpolation-based estimation: we wish to build a risk estimator of the form

$$\hat{R}_{\text{ASE}}(\boldsymbol{\phi}) = \frac{1}{N} \sum\nolimits_{i \in \mathcal{D}_{\text{test}}} g(\boldsymbol{x}_i; \boldsymbol{\phi}), \tag{5.1}$$

where $g(\boldsymbol{x}_i; \boldsymbol{\phi})$ is a flexible surrogate model with parameters $\boldsymbol{\phi}$ that approximates the conditional expected loss, $\mathbb{E}\left[\mathcal{L}(f(\boldsymbol{X}), Y) \mid \boldsymbol{X} = \boldsymbol{x}\right]$.

Though algorithmically quite different, the motivation behind this interpolation-based approach is quite similar to that of variational inference (VI) (Zhang et al., 2018): rather than relying on a sample-based approximation for our risk estimator, we are converting our estimation into the optimization of a functional approximation. Though, as with VI, this comes at the cost of weaker theoretical asymptotic guarantees compared with MC estimation (see Section 5.4), it allows us to generalize more effectively when only a small number of labels are available. This trade-off is typically worthwhile in the small sample size regime of active testing.

To decide what form g should take, we first note that the theoretically optimal g^* – in terms of mean squared error (MSE) for the risk – is simply the expected loss above, analogously to the optimal proposal for LURE. While this is itself obviously intractable, we can learn an effective g by trying to approximate this optimal g^*. Though one could directly approximate this using a generic regressor, such as a neural network, this would disregard that we already know the loss function $\mathcal{L}$, can evaluate f, and know how both are combined to give the optimal g^*. In fact, the only unknown is the conditional expectation $Y \mid \boldsymbol{X} = \boldsymbol{x}$ for the test distribution.

We therefore introduce an auxiliary model $\pi(y \mid \boldsymbol{x}; \boldsymbol{\phi})$ of only this conditional.[2] Combining π, f, and $\mathcal{L}$, we construct g as $g(\boldsymbol{x}; \boldsymbol{\phi}) = \mathbb{E}_{Y \sim \pi(\cdot \mid \boldsymbol{x}; \boldsymbol{\phi})} \left[\mathcal{L}(f(\boldsymbol{x}), Y)\right]$, where the only learnable components of g are the parameters $\boldsymbol{\phi}$ of π. With π introduced, we can finally write the ASE as

$$\hat{R}_{\text{ASE}}(\boldsymbol{\phi}) = \frac{1}{N} \sum\nolimits_{i \in \mathcal{D}_{\text{test}}} \mathbb{E}_{Y \sim \pi(\cdot \mid \boldsymbol{x}_i; \boldsymbol{\phi})} \left[\mathcal{L}(f(\boldsymbol{x}_i), Y)\right], \tag{5.2}$$

where we emphasize that the expectation over $Y \mid \boldsymbol{X} = \boldsymbol{x}_i$ is now with respect to π and not the unknown true label conditional. Note that the fact that the ASE includes an explicit model of the label noise is a decisive advantage over prior work (see Section 5.7). For classification, we can now directly evaluate the expectation over the labels as a weighted sum over the predictive probabilities $\sum_y \pi(y \mid \boldsymbol{x}_i; \boldsymbol{\phi}) \mathcal{L}(f(\boldsymbol{x}_i), y)$. For regression, closed-form solutions of the expectation exist for some losses, and we can always compute unbiased and accurate MC estimates using cheap samples from π.

Assembling g from $\mathcal{L}$, f, and π will typically be advantageous to modeling g directly: a direct model would need to learn not only about predictions of f, but also how they relate to the test distribution. In contrast, learning models $\pi(y \mid \boldsymbol{x}; \boldsymbol{\phi})$ of conditional outcomes is standard procedure. This also makes it easy to include other prior information. For example, initializing π on the *training* data can sometimes give the ASE a useful estimate of the risk before observing any test data at all. Lastly, the modular construction of g allows us to quickly adapt the ASE to different models or loss functions.

Often, f, like π, will be a probabilistic model of outcomes. However, it will generally be an inappropriate choice for π, such that it is important to always learn a separate auxiliary model. An argument similar to that in Chapter 4 applies: we are essentially using π to predict the errors of f, and we should therefore aim to decorrelate the errors of π from those of f by training a separate π. Further, we may want to choose π from a different model class than f, e.g. to allow for

[2] In Chapter 4, we have called π the surrogate model. In this chapter, we call π the auxiliary model to delineate it from the surrogate for the expected loss g. Further, our decision to denote the parameters of g with $\boldsymbol{\phi}$ instead of $\boldsymbol{\theta}$ as in previous chapters will become clear later on.

Algorithm 2: Adaptive Refinement of ASEs

Input: Test pool $\{\boldsymbol{x}_i\}_{i \in \mathcal{D}_{\text{test}}}$, acquisition strategy a, target model f, initial auxiliary model π_0

Initialize $\mathcal{O} = \emptyset$ and $\mathcal{U} = \mathcal{D}_{\text{test}}$
for $m = 1$ *to* M **do**
  Select $i_m = a(\pi_{m-1}, f, \{\boldsymbol{x}_i\}_{i \in \mathcal{D}_{\text{test}}}, \{y_i\}_{i \in \mathcal{O}})$
  Acquire label $y_{i_m} \sim p(Y \mid \boldsymbol{x}_{i_m})$
  Update $\mathcal{O} \leftarrow \mathcal{O} \cup i_m$ and $\mathcal{U} \leftarrow \mathcal{U} \setminus i_m$
  Update auxiliary model π_m (e.g. by retraining)
end
return $\hat{r}_{\text{ASE}}$ as per Eq. (5.2) with $\pi = \pi_M$

meaningful incorporation of uncertainties. Lastly, while f is by construction fixed, we can adapt π during evaluation to reflect our updated knowledge of the test label distribution, as we explain next.

5.2.2 Adaptive Refinement of ASEs

ASEs are adaptive: they iteratively improve their estimate by updating the surrogate as new labels are acquired. A crucial component to them doing this effectively is that they actively select the labels to acquire at each iteration using an acquisition strategy. As such, they can be thought of as performing estimation via active learning of the auxiliary model π, noting that the estimate they produce is, in turn, fully defined by this auxiliary model and the pool set.

To define this adaptive refinement process more formally, we denote with $\mathcal{U}$ and $\mathcal{O}$ the sets of unobserved and observed points in the test pool. These are initialized as $\mathcal{U} = \mathcal{D}_{\text{test}}$ and $\mathcal{O} = \emptyset$. At each subsequent iteration m of our adaptive refinement process, an acquisition strategy a selects an index $i_m \in \mathcal{U}$ of an unlabeled datapoint, for which we then acquire a label y_{i_m}. We remove i_m from $\mathcal{U}$, and add it to $\mathcal{O}$. Then, we obtain an improved ASE by retraining π, e.g. on the total observed test data $\{(x_i, y_i) : i \in \mathcal{O}\}$; this retraining does not necessarily need to be performed at every iteration if doing so would be computationally infeasible. A summary of this approach is given in Algorithm 2.

In Appendix C.4, we show that the computational complexity of ASEs is given by $\mathcal{O}\left(M \cdot (N + |\mathcal{D}_{\text{train}}|)\right)$, assuming an initial training set of size $|\mathcal{D}_{\text{train}}|$ for

the surrogate. This is more expensive than a naive MC estimate but identical to the computational complexity of the LURE-based active testing approach from Chapter 4.

In ASEs, our acquisition strategy takes the current auxiliary model π_m, the original model f, the pool of samples, and all previous evaluations as input, i.e. we have $i_m = a(\pi_{m-1}, f, \{\boldsymbol{x}_i\}_{i \in \mathcal{D}_{\text{test}}}, \{y_i\}_{i \in \mathcal{O}})$.

Following our discussion of bias in active learning (Farquhar et al., 2021) in Section 2.3, we use a standard unweighted empirical risk objective when iteratively retraining the auxiliary model and do not attempt to correct any biases introduced by the active acquisition of the training datapoints for the auxiliary model.

A good acquisition strategy selects points such that the expected ASE error after M iterations is minimal (Imberg et al., 2020). While ASEs can be used with arbitrary acquisition strategies, we next introduce a novel acquisition strategy that we specifically design for use with ASEs.

5.3 The XWED Acquisition Function

It is important we tailor our acquisition strategy to the problem of active surrogate estimation, rather than apply more generic active learning approaches. We need to acquire labels such that we produce the best ASE estimate given a limited labeling budget. To do this, we propose a novel acquisition function called the E**x**pected **We**ighted **D**isagreement, or *XWED* for short, that is designed specifically to target improvements of the ASE estimate itself.

In line with acquisition strategies in active learning, we now require that π takes the form $\pi(y \mid \boldsymbol{x}; \boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\Theta} \sim \pi(\cdot; \boldsymbol{\phi})} [\pi(y \mid \boldsymbol{x}, \boldsymbol{\Theta}; \boldsymbol{\phi})]$, where $\boldsymbol{\Theta}$ represents some model parameters we wish to learn about. In a BNN, $\boldsymbol{\Theta}$ is the weights and biases of the network, while $\boldsymbol{\phi}$ corresponds to the parameters of the posterior approximation given by previous observations (such that it contains all information from previous data). To avoid clutter, we omit $\boldsymbol{\phi}$ in the following presentation of acquisition functions.

In Section 2.3, we have already introduced BALD (Houlsby et al., 2011), an established acquisition strategy for active learning that acquires points with high

epistemic uncertainty (Der Kiureghian and Ditlevsen, 2009; Kendall and Gal, 2017). For a predictive model π as above, BALD can be calculated as the disagreement between the total predictive entropy and the expected entropies over posterior draws,

$$\mathrm{BALD}(\boldsymbol{x}) = \mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x})} \left[-\log \pi(Y \mid \boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{\Theta} \sim \pi(\cdot)} \left[\mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x}, \boldsymbol{\Theta})} \left[-\log \pi(Y \mid \boldsymbol{x}, \boldsymbol{\Theta}) \right] \right]. \tag{5.3}$$

However, BALD, and other acquisition functions from active learning, are ill-suited to ASEs because they do not consider the effect the uncertainties in π have on the ASE estimate. In particular, ASEs depend on the loss $\mathcal{L}(y, f(\boldsymbol{x}))$ of the original model f at the pool samples, but neither the form of the loss function $\mathcal{L}$ nor the original model f are considered in BALD acquisition.

Acquisition for ASEs should instead target points that are both informative about $\boldsymbol{\Theta}$ *and* which we expect will contribute significantly to our final risk estimate because they are high loss. With XWED, we propose an ad-hoc modification of BALD that *weights* the disagreement terms for particular inputs $\boldsymbol{x}$ under the surrogate using the corresponding loss $\mathcal{L}(Y, f(\boldsymbol{x}))$ of the original model f:

$$\begin{aligned} \mathrm{XWED}(\boldsymbol{x}) = &\mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x})} \left[-\mathcal{L}(Y, f(\boldsymbol{x})) \log \pi(Y \mid \boldsymbol{x}) \right] \\ &- \mathbb{E}_{\boldsymbol{\Theta} \sim \pi(\cdot)} \left[\mathbb{E}_{Y \sim \pi(\cdot|\boldsymbol{x}, \boldsymbol{\Theta})} \left[-\mathcal{L}(Y, f(\boldsymbol{x})) \log \pi(Y \mid \boldsymbol{x}, \boldsymbol{\Theta}) \right] \right]. \end{aligned} \tag{5.4}$$

Unlike acquisition with BALD, XWED acquires points with large epistemic uncertainty (Der Kiureghian and Ditlevsen, 2009; Kendall and Gal, 2017) in the auxiliary π about the outcomes only if they are expected to be high loss, and thus relevant to the estimate $\hat{R}_{\mathrm{ASE}}$. In preliminary experiments, we have also considered acquisition with a more straightforward loss-weighted epistemic uncertainty, $\mathbb{E}\left[\mathcal{L}(Y, f(\boldsymbol{x}))\right] \cdot \mathrm{BALD}(\boldsymbol{x})$, with separate expectations over Y for the two terms. With XWED, we instead share expectations over Y between loss and epistemic uncertainty, which we have found to perform better in practice, presumably, because this allows capturing co-variance in Y between the two terms.

To convert this acquisition function to an acquisition strategy, we simply choose the point for which the XWED score is maximal: $a(\pi_{m-1}, f, \{\boldsymbol{x}_i\}_{i \in \mathcal{D}_{\mathrm{test}}}, \{y_i\}_{i \in \mathcal{O}}) =$

$\arg\max_{i \in \mathcal{U}} \text{XWED}(\boldsymbol{x}_i)$. As we show in Section 5.6, acquisition from XWED allows for significant improvements compared to BALD and other strategies.

XWED is compatible with a variety of model classes for π, such as Bayesian Neural Networks (BNNs) (MacKay, 1992a; Gal et al., 2017) and deep ensembles (Lakshminarayanan et al., 2017). In Appendix C.3.2, we give a worked through example for the practical computation of XWED when the main model f is a neural network and the surrogate π a deep ensemble, both trained for classification. We emphasize that XWED is an acquisition strategy explicitly designed for active testing; it is not even defined in a traditional active learning context, where the distinction between a fixed model f and the adaptive auxiliary model π does not exist.

5.4 Theoretical Analysis of the ASE Error

We next provide a theoretical analysis of ASEs to understand individual contributions to their error. Specifically, we will break down the errors of ASEs into (I) error from using a finite evaluation pool and (II) error from imperfection of the surrogate, further breaking the latter down as (IIA) error from any limited expressivity in our surrogate class and (IIB) error from imperfect training.

To start, let $\boldsymbol{\Phi}$ be the final parameters obtained from the stochastic training procedure of the surrogate g, such that our learned surrogate is $g(\cdot; \boldsymbol{\Phi})$. Here, $\boldsymbol{\Phi}$ is a random variable with the stochasticity originating from (i) sampling the test pool itself, (ii) any stochasticity in the acquisition strategy, (iii) randomness in the acquired labels, and (iv) stochasticity in the optimization process of the surrogate.

Next, let $\boldsymbol{X}_0 \sim p(\boldsymbol{x})$ be a hypothetical additional input sample that is independent from the test pool. We can use this to define $G(\boldsymbol{\phi}) = \mathbb{E}\left[g(\boldsymbol{X}_0; \boldsymbol{\phi})\right]$ as the true expectation with respect to the input distribution of a surrogate with parameters $\boldsymbol{\phi}$. In turn, we can define $\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} |G(\boldsymbol{\phi}) - r|$ as the parameters which give the best approximation of the true expected loss $\mathbb{E}\left[\mathcal{L}(f(\boldsymbol{X}), Y) \mid \boldsymbol{X} = \boldsymbol{x}\right]$. Note here that $\boldsymbol{\phi}^*$ is not random as it depends only on the fixed function class of g and the fixed true risk $r = \mathbb{E}\left[\mathcal{L}(f(\boldsymbol{X}), Y)\right]$.

Denoting our test pool as $\boldsymbol{P} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$, where $\boldsymbol{X}_i \sim p(\boldsymbol{x})$ are independently sampled, we can write $\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P})$ to make the dependency of the ASE estimator (Eq. (5.2)) on $\boldsymbol{P}$ and $\boldsymbol{\Phi}$ explicit. By using the triangle inequality, we can then break down the error $\left\|\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P}) - r\right\|$ to the true model risk r as

$$\begin{aligned}\left\|\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P}) - r\right\| &= \left\|\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P}) - G(\boldsymbol{\Phi}) + G(\boldsymbol{\Phi}) - G(\boldsymbol{\phi}^*) + G(\boldsymbol{\phi}^*) - r\right\| \\ &\leq \left\|\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P}) - G(\boldsymbol{\Phi})\right\| + \|G(\boldsymbol{\Phi}) - r\| \\ &\leq \left\|\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P}) - G(\boldsymbol{\Phi})\right\| + |G(\boldsymbol{\phi}^*) - r| + \|G(\boldsymbol{\Phi}) - G(\boldsymbol{\phi}^*)\|,\end{aligned} \tag{5.5}$$

where $\|\cdot\|$ is any valid norm, and we note that $G(\boldsymbol{\phi})$ is independent of $\boldsymbol{X}_0$ and $G(\boldsymbol{\phi}^*)$ is deterministic. We now consider each of the terms in Eq. (5.5):

$(I) = \|\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P}) - G(\boldsymbol{\Phi})\|$ is the error that originates from only evaluating the surrogate on a finite set of pool of points. As per standard results for MC estimation, this will generally decrease as $\mathcal{O}(1/\sqrt{N})$, such that it will be small for large test pools. Note here that $\hat{R}_{\text{ASE}}(\boldsymbol{\Phi}, \boldsymbol{P})$ is typically not an unbiased estimator for $G(\boldsymbol{\Phi})$ as the set of input points available in the pool can influence the distribution of $\boldsymbol{\Phi}$. However, this effect will typically be very small and the bias will diminish as N increases.

We can also, if desired, eliminate it completely by evaluating g over a separate test pool than that used during the acquisition of test labels and updating of g. This variation on the standard ASE estimator can also be useful when the pool is impractically large for performing label acquisition, as we can evaluate the final learned g on a larger pool than used during its training.

$(II) = \|G(\boldsymbol{\Phi}) - r\|$ represents the error from the imperfection of the surrogate. For the squared norm, it equals $\mathbb{E}_{\Phi}[(\mathbb{E}_{X_0}[g(\boldsymbol{X}_0; \Phi)] - r)^2]$, such that it is the expected squared difference (over surrogate parameters) between the surrogate's expectation over inputs and the true risk. It can itself be further broken down into terms (IIA) and (IIB) as described below. Note that this term demonstrates the critical dependency of ASE's performance on the regressional performance of the surrogate itself.

$(IIA) = |G(\boldsymbol{\phi}^*) - r|$ gives the error from limits in the choice of the surrogate function class. It measures the difference between g^* and $g(\cdot; \boldsymbol{\phi}^*)$. Whenever our

surrogate function class contains the true expected loss, this term will be exactly zero. Even when this is not the case, it should generally be negligible for sensible choices of the surrogate function class. For example, if π is a deep neural network, the limiting factor will almost always be the training error from using finite data (i.e. term (IIB) below), rather than the theoretical expressivity of the network.

$(IIB) = \|G(\mathbf{\Phi}) - G(\boldsymbol{\phi}^*)\|$ can be thought of as the error originating from the training of the surrogate: it is the norm of the difference between the true expectation of the surrogate we have learned and the true expectation of the best possible surrogate we could have learned.

5.4.1 Discussion and Empirical Analysis

In practice, (IIB) will typically dominate the ASE error. If term (I) is significant, the pool size is a limiting factor, which can only be the case when our ASE estimation has been successful (assuming $M \ll N$), as its implies we have a highly accurate estimate of the full empirical test risk $\hat{r}$. If term (IIA) is significant, this means we are using an inappropriately weak model class for our surrogate.

To check these assertions hold in practice, we empirically estimate (I) and (IIA) for the experiments in Section 5.6.1 at $M = 1$: we find that they respectively make up only 0.00013% and 0.031% of the full error; thus (IIB) is completely dominant here.

Further characterizing the dominating term (IIB) in the fully general setting is unfortunately not feasible: it equates to characterizing the error of a regression, which necessarily can only be done by considering a particular class of surrogate. However, for a particular choice of surrogate, existing results from learning theory could be applied to further categorize its convergence. Here work on the convergence of models in active *learning* (Hanneke, 2011a; Hanneke, 2011b; Sabato and Munos, 2014; Chaudhuri et al., 2015; Raj and Bach, 2022) will be particularly appropriate, noting that these all make specific assumptions about the model and acquisition strategy. More specifically, for Gaussian process surrogates and acquisition strategies from Bayesian quadrature, the results of Kanagawa and

Hennig (2019) could be applied. On the other hand, for surrogates based on deep learning architectures, one could use results from the convergence of stochastic gradient descent methods (Moulines and Bach, 2011), which typically give rates of $\mathcal{O}(1/\sqrt{M})$ under appropriate assumptions. These will require unbiased gradients in the optimization of the objective function, which we can achieve for ASEs with stochastic acquisition and the de-biasing scheme of Farquhar et al. (2021).

In summary, for large pool sizes and appropriate surrogate classes, (IIB) will be the dominant contributor to the error, noting that the inequality in Eq. (5.5) becomes tight as one of the errors dominates. As such, the convergence of ASEs essentially reduces to the convergence of the surrogate, and thus, by extension, the convergence of the auxiliary model. Moreover, if we can ensure that this auxiliary model convergences to the true conditional output distribution as the number of label acquisitions M increases (this itself implies that our surrogate function class is sufficient powerful and so $(IIA) = 0$), we can thus ensure that the ASE estimator itself converges, noting that $M \to \infty$ also predicates $N \to \infty$ (as $N > M$ by construction), such that $(I) \to 0$.

5.5 Related Work

We refer to Section 2.3 for a general overview of strategies for label-efficient machine learning and to Section 4.4 for work directly related to label-efficient model evaluation.

Surrogates The use of surrogate models to replace expensive black-box functions is pervasive throughout much of the sciences (Qian et al., 2005; Kleijnen, 2009; Cozad et al., 2014). Bayesian optimisation (BO) (Shahriari et al., 2015; Brochu et al., 2010) and Bayesian quadrature (BQ) (O'Hagan, 1991; Rasmussen and Ghahramani, 2003; Briol et al., 2019) are prominent examples of surrogate-based approaches that actively select samples. BO iteratively finds the maximum of an unknown function, using a surrogate model and ideas similar to active learning to select sample locations. Unlike active learning, BO does not focus on a globally accurate

surrogate. BQ obtains a Bayesian posterior for the integration of a Gaussian process (GP) (Rasmussen, 2003) surrogate against a known data density $p(\boldsymbol{x})$. Despite biased estimates, BQ can drastically outperform MC in low dimensions. However, BQ does not scale well to high dimensions, where, often, GP surrogates are not appropriate and explicit densities $p(\boldsymbol{x})$ are unknown and hard to approximate.

Model Evaluation without Labels Similar to our experiments in Section 5.6.3, Guerra et al. (2008), Redyuk et al. (2019), Corneanu et al. (2020), Deng and Zheng (2021), Deng et al. (2021), Sun et al. (2021), and Talagala et al. (2022) study the evaluation of model test performance without test labels or even without test sets. However, unlike ASEs, these works are based around performing meta analysis and require access to multiple related datasets (for which test labels are available). They rely on regressions from dataset/model summaries to performance scores and cannot be used in the setting considered here, where only one dataset is available. Similarly, meta-learning for hyperparameter optimization, e.g. Eggensperger et al. (2015), Vanschoren (2018), Eggensperger et al. (2018), Yu and Zhu (2020), and He et al. (2021), interpolates test performance across a variety of datasets *and* model hyperparameters.

5.6 Experiments

We next study the performance of ASEs for active testing in comparison to relevant baselines. Concretely, we compare to naive MC subsampling, cf. Eq. (4.2), and the active testing approach introduced in Chapter 4, that we refer to as LURE-based active testing in the following for clarity. As a general and common application, we consider the important problem of actively evaluating deep neural networks trained on high-dimensional image datasets.

We give full details on the experiments in the appendix. In particular, Appendix C.1 and Appendix C.2 contain additional results and figures, and Appendix C.3 gives further details on the computation of XWED and the baselines.

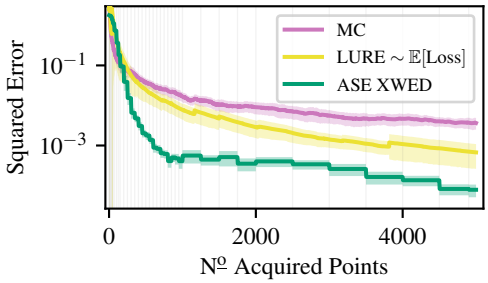


Figure 5.1: ASE significantly improves over LURE and naive MC in terms of squared error for the distribution shift experiment. Shown are mean errors and the shading is two std. errors over 100 runs. We retrain π at steps marked with grey lines.

5.6.1 Evaluation with Distribution Shift

We first explore model evaluation in a challenging distribution shift scenario. Concretely, we are given a fixed model trained on 2000 digits of MNIST (LeCun et al., 1998), where all sevens have been removed from the training set. We then estimate the model risk on a test set with all classes present in their normal proportions. The test set has size $N = 60000$ but we can only afford to acquire labels for a small subset. Such distribution shift settings are relevant in practice *and* challenging for active testing: both ASE- and LURE-based active testing needs to approximate the expected loss, and this is particularly hard under distribution shift, where the initial π has yet to learn about the test distribution.

Following our experiments in Chapter 4, we again use radial Bayesian Neural Networks (Farquhar et al., 2020) for f and π, with π initialized on the training set and regularly retrained during evaluation. For LURE, we acquire labels by sampling from Eq. (4.6). For the ASE, we use the XWED acquisition strategy given in Eq. (5.4). Both ASE and LURE use the same auxiliary model class π, but arrive at different network weights, $\boldsymbol{\phi}_{\text{ASE}}$ and $\boldsymbol{\phi}_{\text{LURE}}$, because their distinct acquisition strategies lead to different test observations used for the re-training of π.

Main Result Fig. 5.1 shows the MSE for ASE and baselines when estimating the unknown test pool risk. Note here that we are careful to use the best possible setup of the LURE baseline to construct a challenging benchmark, as shown in the next section. We see that the ASE improves on all baselines when used with

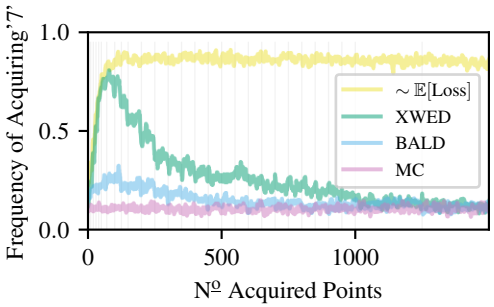


Figure 5.2: Proportion of acquired points which have class "7". Results averaged over 100 runs.

the XWED acquisition strategy. Further, LURE can struggle with variance, with errors of individual runs occasionally spiking. This occurs when there is a significant mismatch between observed loss and proposal, e.g. when an acquired point is ambiguous or mislabeled. This does not happen for ASEs, as we do not weight true observations against predictions in the estimate.

5.6.2 Acquisition Strategies for ASE and LURE

We next perform an ablation that investigates different acquisition strategies besides XWED in the distribution shift scenario. For one, we select points with maximal BALD score, Eq. (5.3), to study if our custom XWED acquisition provides benefits over the popular active learning acquisition strategy. Further, we acquire points by sampling from the expected loss; this is the exact same acquisition strategy used for LURE-based active testing, cf. Eq. (4.6). See Appendix C.3.1 for further details on how we compute these baselines.

Fig. 5.2 visualizes how often the acquisition functions acquire the missing class as testing progresses, and Fig. 5.3 (a) shows that ASE is able to outperform naive MC for *all* acquisition strategies, but there are noticeable variations in their performance. The XWED acquisition, that we have proposed specifically with ASEs in mind, provides the best performance. BALD acquisition does not target those points that lead to the largest improvements in the estimate, which leads to a much slower decrease in ASE error early on, though it does catch up with XWED later. Expected loss acquisition – which is optimal for LURE – will sample the missing class (where loss is high) almost exclusively, which initially works well but later

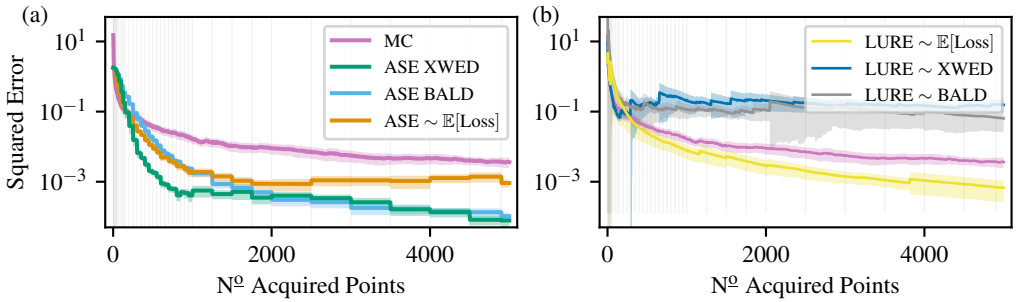


Figure 5.3: Different acquisition strategies for ASEs and LURE. (a) While XWED performs best, ASE outperforms MC for all investigated acquisition functions. (b) LURE suffers high variance for all acquisition strategies except expected loss. Shown are mean errors, shading is two std. errors over 100 runs. We retrain π at steps marked with grey lines. The "$\sim$"-symbol indicates stochastic acquisition.

leads to π failing to improve on the other classes. In contrast, XWED focuses on the missing class early on but – once the loss is well characterized for the missing class – explores other areas as well. This behavior is preferable over the baselines, and we continue this discussion in Appendix C.1.1.

In Fig. 5.3 (b), we explore how LURE behaves under different acquisition strategies, such as sampling from XWED or BALD scores. We find that LURE is noticeably more sensitive to the acquisition strategy than ASE: both XWED and BALD acquisition lead to high-variance estimates that no longer outperform the naive MC baseline. It seems one cannot expect acquisition strategies other than expected loss to perform well with LURE. In contrast, ASEs can directly apply any acquisition scheme without introducing such sensitivity because the acquisition does not affect which points are included in the estimate or how they are weighted.

We provide additional results in the appendix: Appendix C.1.2 shows that adaptive improvement of the acquisition function is necessary for label-efficient active testing in the distribution shift scenario, and, in Appendix C.1.3, we study additional choices for ASE acquisition strategies (which do not outperform XWED).

5.6.3 Evaluation of CNNs for Classification

Next, we study active testing of ResNets on more complex image classification datasets. Following the setup of Section 4.5.3, we again study the efficient evaluation

of a Resnet-18 (He et al., 2016) on Fashion-MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky, Hinton, et al., 2009), and of a WideResNet (Zagoruyko and Komodakis, 2016) on CIFAR-100. In each case, we train the model to be evaluated, on a training set containing 40 000 points, and then use an evaluation pool of size $N = 2000$. As a ground truth, we compare to the empirical risk on a held out set of 20 000 points. For π, we use deep ensembles (Lakshminarayanan et al., 2017) of the original ResNet models.

In this experiment, there is no distribution shift between the training data for f and the data on which we evaluate it. At the same time, the training set is large and, therefore, a lot of prior information is available to train the initial π. Further, the number of test acquisitions is comparatively small ($M = 50$) and unlikely to affect π meaningfully, given the models have already seen 40 000 points from the training data. Indeed, we find that updating π with this additional test data makes no noticeable difference to performance. To minimize computational costs, we therefore report results with π fixed throughout this experiment, such that, here, π is equal for ASE- and LURE-based active testing. A constant π leads to a constant LURE proposal (making the method more akin to IS than AIS) and constant prediction for ASE.

Fig. 5.4 shows that ASEs perform effectively in this setting, significantly outperforming LURE and naive MC for all datasets. We also display the error from the finite pool size, by comparing the empirical risk of all N unobserved test points against our additional ground truth set and see that ASE consistently performs similarly to this pool limit. This suggests that the pool size N may be the limit of the performance for ASEs here rather than the quality of the surrogate.

This experiment produces a rather unusual result: as π is fixed throughout, the ASE actually does not make use of *any* information from the test labels. Its accurate estimates of the test risk are entirely due to information from the training data. Note that LURE here has access to the same π as ASE in addition to up to 50 test labels. Nevertheless, the surrogate prediction of the ASE at all pool points

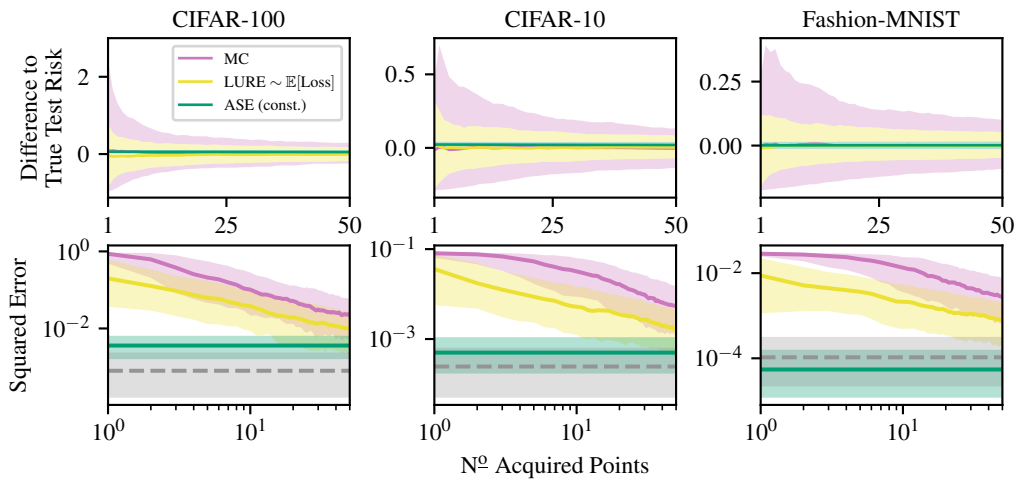


Figure 5.4: Active testing of ResNets on CIFAR-100, CIFAR-10, and FashionMNIST without retraining the auxiliary model π. Despite not observing any test labels, ASE again improves upon LURE. We display medians over 1000 random test sets and 10/90 % (top) and 25/75 % (bottom) quantiles (top: too small to be visible for ASE). In grey, we display a lower bound on the error from the finite pool.

leads to much lower error than a LURE estimate. This "zero-shot" setting opens up interesting avenues for model evaluation when there are no test labels available at all.

However, in practice, one may not know if the test data follow the same distribution as the training data. Fig. C.1 (a) shows that both LURE and ASE perform poorly when not retraining π in the distribution shift setting for any labeling budget M. We therefore generally suggest regular adaptation of π for both ASE- and LURE-based active testing approaches. And even without distribution shift, the information from test labels will always eventually become relevant as M increases.

ASEs are compatible with arbitrary loss functions, cf. Eq. (5.2). For example, for 0-1-Loss, ASEs can be used to estimate the accuracy of the main model. In Appendix C.1.4, we demonstrate that ASEs continue to outperform all other baselines for the task of accuracy estimation.

In our theoretical analysis in Section 5.4, we argue that, in most practical settings, the error originating from the surrogate will dominate the ASE error. In Appendix C.1.5, we reduce the size of the training set to 10 000. This makes the problem more challenging for ASEs because it reduces the quality of the surrogate model. Nevertheless, we find that ASEs maintain their significant advantage over the baselines.

5.7 Discussion

In the previous section, we have seen ASEs outperform MC-based alternatives in practice. Here, we discuss advantages of ASEs that are important to their practical success.

Surrogate Updates For AIS, improving the proposal distribution only improves future acquisitions, but does not retrospectively improve weights attached to earlier observations. While there are techniques for reducing the variance of the weights in the AIS literature (Bugallo et al., 2017; Owen and Zhou, 2000), it is unclear how to apply these to adaptive LURE. This means that bad initial acquisition weights have a lasting influence on the variance of AIS. In contrast, ASEs make better retrospective use of information: the iterative refinement of g leads to updated predictions at all pool points.

Noisy Labels ASEs have a further advantage when label distributions are *noisy*, i.e. observations y are stochastic even for the true label conditional $p(y \mid \boldsymbol{x})$. This is common in practice, because of ambiguous inputs near class-boundaries, mislabeled examples, or measurement error. ASEs directly model the noisy outcome conditionals using π. If π is accurate, the ASE error does not increase due to noisy labels, and the ASE can interpolate beyond the noise. In contrast, AIS suffers badly: noisy observations form a direct part of the estimate, leading to increased variance; while the acquisition in Eq. (4.6) models the noise, this only affects weights, and noisy loss observations directly enter the estimate in Eq. (4.3). Chen and Choe (2019) provide a detailed analysis of how stochastic observations affect IS estimators negatively.

5.8 Conclusions

We have introduced Active Surrogate Estimators (ASEs), a surrogate-based method for label-efficient model evaluation. The surrogate is actively learned, and we introduced the XWED acquisition function to directly account for the risk prediction

task. Our experiments show that ASEs and XWED acquisition give state-of-the-art performance for efficient evaluation of deep image classifiers.

This concludes our investigation into label-efficient model evaluation. In the next chapter, we turn to a particularly exciting method for data-efficient deep learning: in-context learning in large language models.

6

In-Context Learning of Label Relationships in LLMs

Contents

Authorship Statement

The work presented in this chapter has previously been published as Kossen et al. (2024). Section 1.3 gives a detailed account of the contributions my collaborators have made to this work.

6.1 Introduction

As we have previously discussed in Section 2.2.2, Brown et al. (2020) have shown that Large Language Models (LLMs) (Radford et al., 2019; Chowdhery et al., 2022; Hoffmann et al., 2022; Zhang et al., 2022a) can perform so-called *in-context learning* (ICL) of supervised tasks. In contrast to standard in-weights learning, e.g. gradient-based finetuning of model parameters, ICL requires no parameter updates. Instead, examples of the input–label relationship of the downstream task are simply prepended to the query for which the LLM predicts. This is sometimes also referred to as *few-shot ICL* to differentiate from other ICL variants that do not use example demonstrations (Liu et al., 2023a). ICL can be highly data-efficient for some applications: given the breadth of world knowledge contained in pre-trained LLMs, the number of input demonstrations required for successful ICL can often be significantly smaller than for conventional model training. Few-shot ICL is widely used, e.g. in all LLM publications cited above, to improve predictions across a variety

of established NLP tasks, such as sentiment or document classification, question answering, or natural language inference. However, there is currently no consensus on *why* ICL improves predictions, with prior work presenting a large variety of often contradictory perspectives. For example, Brown et al. (2020) highlight similarities between the behavior of ICL and finetuning of LLMs, such as improvements with model size and number of examples. Since then, some have argued that ICL works because it implements general-purpose learning algorithms such as Bayesian inference or gradient descent (Xie et al., 2022; Huszár, 2023; Hahn and Goyal, 2023; Jiang, 2023; Zhang et al., 2023; Von Oswald et al., 2023; Akyürek et al., 2023; Han et al., 2023). In contrast, others have highlighted practical shortcomings of ICL, suggesting ICL does not really "learn" from examples in the way one expects (Liu et al., 2022; Lu et al., 2022; Zhao et al., 2021; Chen et al., 2023; Agrawal et al., 2023; Chang and Jia, 2022; Razeghi et al., 2022; Li and Qiu, 2023; Wei et al., 2023). In particular, Min et al. (2022b) claim their "findings suggest that [LLMs] do not learn new tasks at test time" in the sense of "capturing the input-label correspondence". Clearly, these claims, if true, are not compatible with the behavior we would expect from a general-purpose learning algorithm.

In this chapter, we address the pressing need for an improved understanding of how information given in-context affects ICL predictions. Concretely, we formulate a set of questions that encode our beliefs about how an idealized *conventional learning algorithm* should incorporate label information and then study ICL behavior in relation to this concept of a conventional learner. **(1)** Does ICL take the input–label relationship of the in-context examples into account when predicting for test queries? **(2)** Is ICL powerful enough to overcome prediction preferences originating from pre-training? **(3)** Does ICL treat all information provided in-context equally?

To study these questions rigorously, we rephrase them as null hypotheses that we study empirically. Our results yield an improved understanding of the similarities and differences between ICL and idealized conventional learners.[1]

[1]We provide the code to reproduce our results at github.com/jlko/in_context_learning.

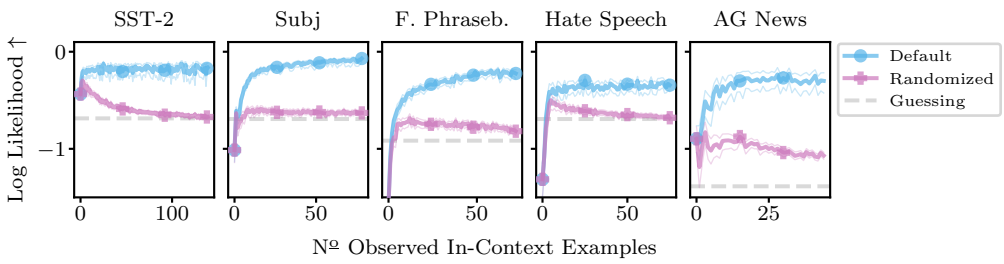


Figure 6.1: ICL predictions generally depend on the conditional label distribution of in-context examples: when in-context labels are randomized, average log likelihoods of label predictions decrease compared to ICL with default labels for LLaMa-2-70B across a variety of tasks. Results averaged over 500 in-context datasets and thin lines are 99 % confidence intervals. See Section 6.5 for details.

Unlike prior work, we study in detail how ICL predictions evolve as an increasing number of examples are provided, from no examples at all up to the maximum possible, across a range of LLMs and tasks. We further show that using probabilistic metrics better highlights the resulting ICL dynamics, often revealing large changes in the confidence of ICL predictions, even when accuracy metrics barely change at all. These measures ensure we obtain a comprehensive picture of ICL behavior.

In our experiments, we first examine if ICL predictions depend on the labels of in-context examples by studying how probabilistic metrics react to randomized in-context labels (Section 6.5, Fig. 6.1). Further, we study ICL on a truly novel task the LLM *cannot* know from pre-training (Section 6.6). Both experiments show that ICL typically considers in-context label relations. We then investigate if ICL is powerful enough to overcome prediction preferences learned from pre-training data (Section 6.7). In our experiments, we find this is typically not the case as ICL performance plateaus if label relations oppose pre-training preference. Further, while additional prompting can improve ICL here, we ultimately do not find prompts that lead to the desired behavior. Finally, we study if ICL treats all information provided in-context equally (Section 6.8). This is important when the context contains multiple different label relationships. By modifying label relations during ICL, we find it does not treat all in-context information equally, and, instead, ICL preferentially makes use of information closer to the query.

In summary, our results suggest a new middle ground regarding the capabilities and limitations of ICL. While ICL can learn from label information, it does so

differently than an idealized learner. Our findings thus contribute to a better understanding of information processing in ICL, which, in turn, is crucial to our ability to deploy LLMs safely and effectively. For example, Bai et al. (2022b) suggest to use ICL for alignment, which relies on ICL being able to sufficiently adjust LLM behavior.

6.2 Background

A large and growing body of prior work studies ICL. We here highlight those studies that are most relevant to our motivation and discuss other related work later in Section 6.9.

Since its introduction by Brown et al. (2020), few-shot ICL has become an integral part of LLM evaluations. For example, many recent publications rely on few-shot ICL tasks, such as the popular HELM benchmark (Liang et al., 2023), to evaluate their LLMs (Chowdhery et al., 2022; Hoffmann et al., 2022).

In the wake of ICL's success, follow up work has speculated if ICL implements a general purpose learning algorithm such as gradient descent (Von Oswald et al., 2023) or Bayesian inference (Xie et al., 2022). This line of work implies that ICL captures not just how much LLMs have learned during pre-training but, rather, how much LLMs have *learned how to learn* novel supervised tasks in-context. However, so far arguments have been largely theoretical, lacking solid experimental evidence in actual LLMs (Zhang et al., 2023; Huszár, 2023; Wies et al., 2023; Jiang, 2023).

Conversely, a variety of studies have highlighted unexpected shortcomings of ICL. For example, ICL can be sensitive to the formatting (Min et al., 2022b) or order (Lu et al., 2022) of the in-context learning examples. Further, LLMs prefer to predict labels that are common in the pre-training data (Zhao et al., 2021), they can predict drastically different for similar prompts (Chen et al., 2023), and they rely on task formulations similar to those observed in the pre-training data (Wu et al., 2023).

In particular, Min et al. (2022b) claim that ICL does not learn label relationships from in-context examples and that ICL "performance drops only marginally when labels in the demonstrations are replaced by random labels". Further, they suggest

that, instead of learning input–label relationships, ICL only works because the model learns about the general label space, the formatting of the examples, and their input distribution. They assert that ICL does "not learn new tasks at test time" and that "ground truth demonstrations are in fact not required" in many common scenarios. In the first part of our evaluation, we will revisit and ultimately disagree with these claims.

6.3 Null Hypotheses on How ICL Incorporates Label Information

We wish to obtain a better understanding of how ICL uses information about the input–label relationship provided in-context. We therefore study to what extent ICL behavior matches our expectations of how a machine learning algorithm *should* behave in an idealized world. To this end, we introduce the concept of a "conventional learning algorithm" as an algorithm that conforms to our intuitions of how an idealized learner will make predictions given data. In particular, we focus on the following intuitions of an idealized learner: (1) it makes use of the labels for learning, (2) it allows the true label relationship to be learned when provided with sufficient data, and (3) it considers all information in the data equally. Note here that some existing approaches may not always conform to some or all of these intuitions, e.g. due to imperfections in training schemes, but our concept of the conventional learning algorithm reflects how we would expect them to behave if our training worked perfectly.

To allow these intuitions to be tested for ICL, we now introduce a series of null hypotheses. Here, we follow the convention that null hypotheses are constructed with the aim of falsifying them. In other words, the following null hypotheses express the *opposite* of our personal beliefs about the capabilities of ICL, and we aim to reject them given empirical evidence. The null hypotheses do not consistently align with or oppose the behavior of the ideal learner. In fact, for the idealized learner, our first null hypothesis should be rejected, while the second and third null

hypotheses should be accepted. Our first hypothesis follows from the notion that conventional learners make use of the conditional distribution of labels given inputs.

***Null Hypothesis 1 (NH1)**: ICL predictions are independent of the conditional label distribution of the examples given in-context.*

In contrast to prior work (Min et al., 2022b), we believe that NH1 should be rejected and that ICL predictions, like those of the idealized learner, generally depend on the label distribution given in-context. In Section 6.5, we revisit and refine the randomized in-context label experiment of Min et al. (2022b): in addition to revising their results for point predictions, we propose the use of *probabilistic metrics* to study if label randomization really does not affect ICL predictive beliefs. Then, in Section 6.6, we study ICL on a novel task that we create and which ICL can *only* solve by learning a novel label relationship that it cannot know from pre-training.

Next, we study *how* ICL incorporates label information. The pre-trained model already contains information towards many NLP tasks: even without any ICL, predictions are significantly better than random guessing. This raises the question of how information given in-context interacts with this *pre-training preference*. In typical applications of conventional learners, we would expect that predictions eventually follow the label relation of the training examples when provided with sufficient data. A learner that does not conform to this is inherently limited in what it can learn. Unlike previously for NH1, we here believe a priori that ICL will *not* follow the behavior of the idealized learner. Because we aim to reject our null hypotheses, we thus phrase NH2 such that its rejection demonstrates ICL diverging from the idealized learner.

***Null Hypothesis 2 (NH2)**: ICL can overcome the zero-shot prediction preferences of the pre-trained model.*

In Section 6.7, we modify in-context label relationships and study how this changes ICL predictions. If NH2 is true, ICL should eventually predict according to any label relation given in-context.

Our last hypothesis relies on the following typical property of idealized learning algorithms: they consider all information in the examples equally and therefore do

not depend on example order. If a dataset contains multiple sources of information about a label relationship, e.g. the dataset itself is a union of multiple datasets, then predictions should not depend on the order in which these are observed. NH3 investigates if this is holds true for ICL as well. As with NH2, we are again skeptical that ICL behavior is close to ideal here and thus frame NH3 such that successful rejection of it shows ICL predictions opposing the idealized learner.

***Null Hypothesis 3 (NH3)**: ICL considers all information given in-context equally and thus its predictions do not depend on example order.*

In Section 6.8, we study non-stationary input distributions for which label relations *change* during ICL. If NH3 is true, ICL predictions should not depend on the order in which we present label relations.

6.4 Experimental Setup & ICL Training Dynamics

We here detail our experimental setup for the subsequent evaluation of few-shot ICL behavior.

Models & Tasks We employ LLMs from the LLaMa-2 (Touvron et al., 2023b), LLaMa (Touvron et al., 2023a), and Falcon (TII, 2023) families due to their strong performance and open source nature. We evaluate on SST-2, Subjective (Subj.), Financial Phrasebank (FP), Hate Speech (HS), AG News (AGN), MQP, MRPC, RTE, and WNLI. We provide citations for all tasks in Appendix D.4.

Context Size We always report few-shot ICL performance across *all possible* numbers of in-context demonstrations, i.e. from zero-shot performance up to the maximum number of examples within the LLMs' input token limit. This is in contrast to prior work, which often evaluates few-shot ICL at only a few context set sizes, and allows us to obtain a comprehensive picture of ICL "training dynamics".

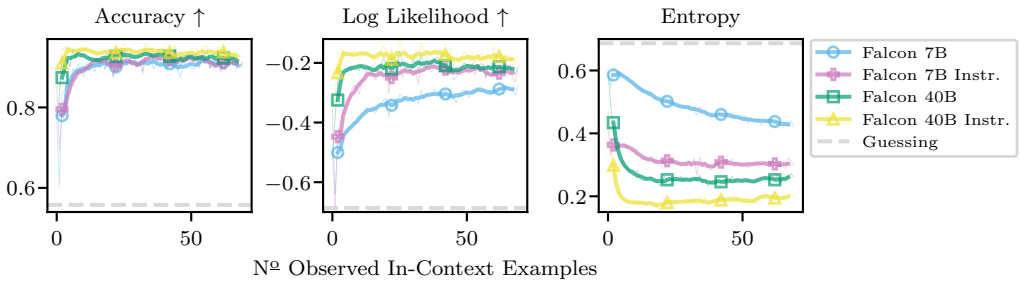


Figure 6.2: Few-shot ICL **training dynamics** in a default label scenario on SST-2. Accuracy (↑) and log likelihood (↑) improve with in-context dataset size, and entropies decrease appropriately. Averages over 500 random subsets, thick lines with moving average (window size 5) for clarity.

Computationally Cheap ICL Evaluations We propose a novel evaluation strategy that obtains ICL predictions at all possible numbers of in-context demonstrations without incurring any additional cost. Concretely, we exploit the fact that each forward pass through the model gives not just a prediction for the next token, but rather, the predicted probabilities for *each* input token (given all preceding tokens). By extracting those token predictions that correspond to labels of in-context examples, we obtain few-shot ICL predictions at all in-context dataset sizes with each forward pass. We refer to Appendix D.2 for a formalization of few-shot ICL and further description of our evaluation strategy.

Evaluation Metrics We evaluate few-shot ICL performance in terms of accuracy (↑) and (average) log likelihood (↑) of label predictions. We also report entropy, which, while not a performance metric, is useful for understanding how much predicted probabilities are spread over classes. We average metrics over sets of in-context examples drawn randomly and without replacement from the training set, and we compare to a guessing baseline that predicts with probabilities equal to class frequencies.

Default Training Dynamics Before modifying label relationships in the following sections, Fig. 6.2 shows standard few-shot ICL training dynamics for Falcon models on SST-2. We observe reasonable behavior for all models: as more in-context examples are observed, accuracies and log likelihoods increase, while entropies

decrease. Notably, log likelihoods and entropies show in-context *learning* more clearly: predicted probabilities continue to improve at larger context sizes, whereas accuracies saturate quickly. Differences between models are more noticeable for probabilistic metrics, too: entropies reveal that larger or instruction-tuned Falcon models predict with higher certainty on SST-2. Similar findings also hold for LLaMa and LLaMA-2 models, for which we provide results in Fig. D.4.

6.5 Do ICL Predictions Depend on In-Context Labels?

We now study the null hypotheses (NH) formulated in Section 6.3, starting with NH1, which states that ICL predictions are independent of the conditional label distribution of the in-context examples. To this end, we first revisit the experiments of Min et al. (2022b), replacing all labels of in-context examples with labels drawn randomly from the training set of the task. If NH1 is true, then accuracy, log-likelihood, and entropy should be identical for the randomized and standard label scenario. We note that, while we believe the results of this experiment are already sufficient to reject NH1, the experiments in Sections 6.6 to 6.8 will provide additional strong evidence for this conclusion.

Observations & Discussion We evaluate NH1 across all our models, tasks, and metrics, computing full ICL training curves as introduced in Section 6.4. Figure 6.1 shows log likelihoods for LLaMa-2-70B for a selection of tasks, and Fig. 6.3 shows all metrics for all Falcon models on SST-2. We observe significant differences in ICL behavior for the default and randomized label scenario. As the context size grows, likelihoods eventually degrade significantly for randomized labels. In Fig. 6.3, we can further see entropies increase when randomizing labels. This is reasonable from a probabilistic learning perspective: as noisy labels are observed, estimates of uncertainty will typically increase. While differences are large for probabilistic

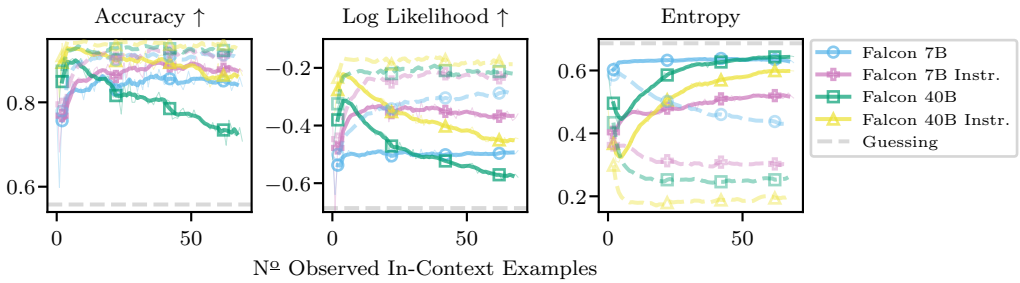


Figure 6.3: Few-shot ICL with **randomized labels** for SST-2: Compared to default ICL behavior (dashed lines), log likelihoods and entropies of the Falcon models degrade when in-context labels are randomized. Accuracies show differences less clearly than probabilistic log likelihood and entropy. Averages over 500 repetitions, thick lines with moving average (window size 5) for clarity.

log likelihood and entropy, they can be harder to spot for accuracy. In Fig. 6.2, only Falcon-40B experiences a sizeable accuracy decrease.[2]

Table 6.1 provides results for label randomization across all our models, tasks, and metrics. It shows the average difference in log likelihoods between the default and randomized labels at the maximum number of demonstrations for each task and model. We gray out entries where ICL on default labels does not outperform the guessing baseline as we are only interested in studying label randomization when ICL works in the first place. When default label performance is better than random (black entries), differences are almost always significantly positive (bold entries), indicating ICL performs worse for randomized labels. ***Based on these results, we reject NH1 that ICL predictions do not depend on the conditional label distribution of in-context examples.***

Notably, Table 6.1 shows that LLaMa-2-70B, our largest and most capable model, always performs worse under label randomization. This suggests the importance of labels in ICL will increase as models become more powerful in the future. However, performance often degrades significantly even for smaller models, although they struggle to reach better than random performance on the entailment tasks MQP, MRPC, RTE, and WNLI. Occasionally, likelihoods improve despite random labels, e.g. for the small Falcon models in Fig. 6.3. Following Pan et al. (2023) and Min et al. (2022b), we attribute this to ICL "recognizing", rather than learning, the task

[2]For LLaMa(-2) models, accuracy decreases more frequently and can approach guessing level, cf. Appendix F in Kossen et al. (2024).

Table 6.1: Average differences between ICL log likelihoods for default and randomized labels. Bold entries indicate differences are statistically significant. We can disregard lightgray entries: for them, default ICL performance is not significantly better than a random guessing baseline. Whenever default ICL outperforms the baseline, ICL almost always performs significantly worse (positive differences) for random labels. Averages over 500 runs at max. context size, standard errors in Table D.2.

Δ Log Likelihood	SST-2	Subj	FP	HS	AGN	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**0.42**	**0.39**	**0.57**	**0.18**	**0.53**	**0.03**	0.02	**0.03**	0.02
LLaMa-2 13B	**0.41**	**0.62**	**0.49**	**0.24**	**0.81**	**0.04**	0.01	**0.06**	0.02
LLaMa-2 70B	**0.51**	**0.53**	**0.57**	**0.34**	**0.80**	**0.29**	**0.04**	**0.22**	**0.18**
Falcon 7B	**0.20**	**0.19**	**0.25**	**0.06**	**0.31**	0.01	0.01	−0.01	0.01
Falcon 7B Instr.	**0.13**	**0.08**	**0.11**	**0.03**	**0.15**	0.03	0.02	−0.00	0.00
Falcon 40B	**0.34**	**0.35**	**0.31**	**0.18**	**0.90**	**0.06**	0.01	0.01	0.02
Falcon 40B Instr.	**0.25**	**0.37**	**0.27**	0.02	**0.77**	**0.06**	0.02	0.02	**0.04**

from the random label demonstrations. However, we note that, even here, there are significant performance gaps to the default scenario. We conclude that label randomization adversely affecting ICL predictions is the rule not the exception.

Discussion of Min et al. (2022b) Lastly, we discuss possible reasons for why Min et al. (2022b) arrive at the conclusion that label randomization only "barely hurts" ICL performance: (1) They do not study probabilistic metrics, which are more sensitive to randomization. (2) They use a fixed ICL dataset size of 16, but effects of random labels increase with growing contexts. (3) Only one model they study has more than 20B parameters (GPT-3), but we observe that larger models react more to randomization. (Pan et al. (2023) also observe this, cf. Section 6.9.) (4) On some tasks, performance for Min et al. (2022b) could be close to random guessing, where label randomization has less of an effect.

6.6 Can ICL Learn Truly Novel Label Relationships?

The results of Section 6.5 show that ICL predictions *do* depend on the label relationship of in-context examples. Here, we explore the *extent* to which ICL can extract label information from the context. Concretely, we study if LLMs can learn *truly novel* label relationships in-context. To do this, we create a task that is

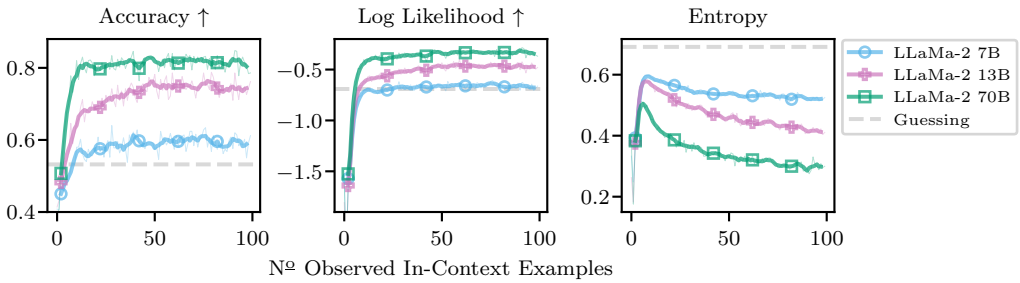


Figure 6.4: Few-shot ICL achieves accuracies significantly better than random guessing on our **novel author identification** task. Thus, LLMs can learn novel label relationships entirely in-context. Averages over 500 runs, thick lines with additional moving average (window size 5) for clarity.

guaranteed to not appear in the pre-training data. The task needs to be distinct from established NLP tasks, for which the pre-training data could be contaminated and for which, often, strong zero-shot performance shows the model has learned the task, perhaps implicitly, during pre-training.

Specifically, we create an authorship identification (Stamatatos, 2009) dataset from private messages between two authors of the original paper. The task is to identify the author corresponding to a given message. As messages stem from private communication, they are guaranteed to not be part of the pre-training corpus. For ICL to succeed here, it needs to learn the novel input–label relationship provided in-context: while the LLM could have some general notion of authorship identification tasks, the specific input–label relationship is definitely novel, as the authors' private writing styles cannot be known to the LLM. We give further details on the task in Appendix D.3.

Observations & Discussion Figure 6.4 shows that ICL with LLaMa-2 succeeds at learning the author identification task. Accuracies and log likelihoods increase, agreeing with expectations about conventional learning. Performance improves with model size, but all models perform better than random. We show results for more LLMs in Fig. D.6: all models, except Falcon-7B-Instruct, achieve better than random performance. We conclude that ***LLMs can learn truly novel tasks in-context***, correctly inferring the label relation from examples. These results

also strongly support our previous rejection of NH1 as, clearly, ICL predictions must depend on labels to learn the novel task.

6.7 Can ICL Overcome Pre-Training Preference?

With NH2, we explore how in-context label information trades off against the LLM's *pre-training preference*, i.e. its zero-shot predictions based on label relationships inferred from pre-training and stored in the model parameters. Often, pre-training preference and in-context label relationships agree: e.g. in Fig. 6.3, performance is high zero-shot and then improves with ICL. To test NH2 if ICL can overcome pre-training preference, we create scenarios where pre-training preference and in-context observations are not aligned. We then study if ICL behavior is compatible with fully overcoming pre-training preference as we would expect from a conventional learner.

Concretely, we use replacement label relationships when constructing the in-context examples. (1) We flip the default labels, e.g. (negative, positive) get mapped to (positive, negative) for SST-2. (2) We study arbitrary labels, e.g. (negative, positive) become (A, B) or (B, A) – we deliberately choose *arbitrary* labels here such that the LLM should not have a significant preference for assigning them to positive or negative. We then evaluate ICL performance for predicting the *replacement label relationship*, e.g. the flipped labels in (1). Note that we rely on the flipped label experiments to evaluate NH2; results on arbitrary labels serve to complete the picture, and we discuss them later.

Observations & Discussion We evaluate NH2 across all our models, tasks, and metrics. Figure 6.5 shows results for a selection of large models and datasets. Evidently, the LLMs can, to some extent, learn to predict the flipped label relationships against pre-training preference. Accuracies on flipped labels reach levels significantly better than random guessing. However, in particular for entropies, there is a consistent gap between the default and flipped label scenarios: ICL predictions on flipped labels are much less certain. Importantly, given the plateauing behavior, this gap is unlikely to disappear with additional in-context observations. (Practically

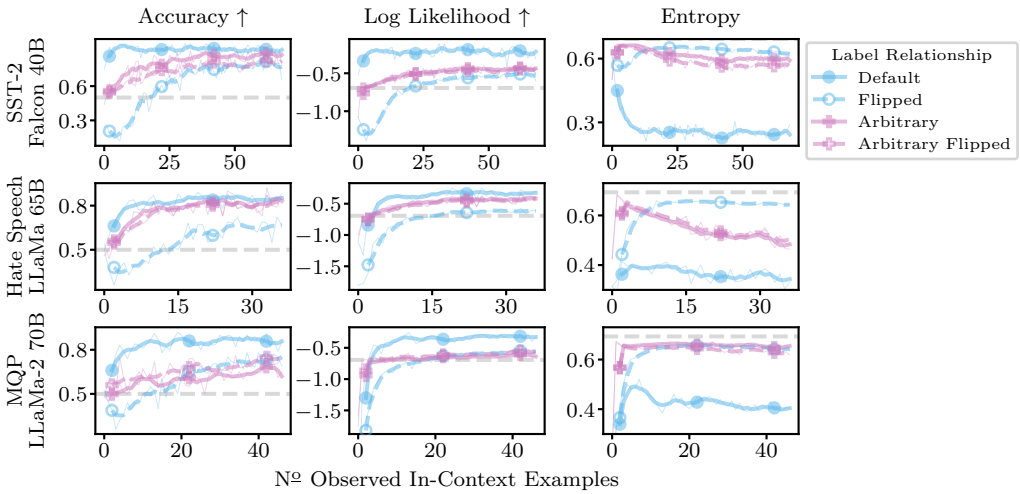


Figure 6.5: Few-shot ICL with **replacement labels** for Falcon-40B on SST-2, LLaMa-65B on Hate Speech, and LLaMa-2-70B on MQP. Table 6.2 and Appendix D.6 contain results for all other models and tasks. ICL achieves better than guessing performance for all label relations and models. However, predictions for flipped labels (dashed blue) plateau at a higher entropies and lower likelihoods than those for the default label relation (solid blue). For arbitrary labels (pink), the model performs similarly for both label directions. Averages over 100 runs and thick lines with moving average (window size 5).

speaking, we cannot actually add any additional examples as input size is maximal already and will deteriorate when exceeding the LLMs' input token limit; this is itself a limitation of ICL compared to conventional learning.) It seems that label relationships inferred from pre-training have a permanent effect that cannot be overcome through in-context observations. This does not agree with conventional learning: predictions on flipped labels should continue to improve as observations continue to contradict pre-training preference.

Crucially, we observe this behavior across models and tasks. Table 6.2 provides a summary of the results, showing differences in entropy between default and flipped label scenarios at maximum context size, highlighting statistical significance in bold and graying out entries where ICL fails on default labels. Across the board, we again observe that predictions on flipped labels plateau: a significant gap between predictions on default and flipped labels remains, even at maximum input size. For the models we study, ***we reject NH2 that ICL can overcome prediction preferences from pre-training.*** Again, the results here strongly support our previous rejection of NH1, as clearly, predictions change for replacement labels.

Table 6.2: Average differences between ICL entropies for default and flipped labels. Bold entries indicate differences are statistically significant. Again, we disregard entries for which default ICL performance is not significantly better than the guessing baseline (lightgray entries). When default ICL outperforms the baseline, ICL entropies are almost always significantly different between scenarios. We average 100 runs, report results at maximum context size, and show standard errors in Table D.3.

Δ Entropy	SST-2	Subj	FP	HS	AGN	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**−0.52**	0.01	**−0.40**	**−0.10**	**−0.75**	−0.00	**0.05**	**−0.03**	−0.01
LLaMa-2 13B	**−0.48**	**−0.06**	**−0.47**	**−0.19**	**−0.95**	**−0.03**	−0.07	**−0.11**	−0.07
LLaMa-2 70B	**−0.17**	**−0.10**	**−0.40**	**−0.26**	**−1.00**	**−0.24**	**−0.15**	**−0.26**	**−0.21**
Falcon 7B	**−0.28**	**−0.12**	−0.02	**−0.06**	**−0.52**	0.00	0.00	−0.00	−0.01
Falcon 7B Instr.	**−0.37**	**−0.07**	**−0.33**	**−0.05**	**−0.22**	−0.02	**0.13**	0.01	0.00
Falcon 40B	**−0.39**	**−0.23**	**−0.42**	**−0.19**	**−0.90**	−0.00	−0.10	−0.02	−0.00
Falcon 40B Instr.	**−0.48**	**−0.16**	**−0.43**	**−0.31**	**−0.92**	**−0.10**	−0.02	**−0.06**	−0.01

Figure 6.5 also shows that for replacement labels (A, B) and (B, A) both directions are similarly easy for ICL to learn. This indicates the LLM truly has not learned a preference for them during pre-training, agreeing with our intuition. Further, learning arbitrary replacement labels is slower than learning from the aligned default labels but faster than learning flipped labels, which does agree with intuitions about the behavior of inductive biases.

In Appendix D.1, we further study if specific prompts, i.e. instructions that inform the LLMs of the flipped labels, can improve ICL predictions. We find that, while some prompts initially can help the model predict on flipped labels, eventually, prompts no longer improve predictions for the selection of models, tasks, and prompts that we study.

6.8 How Does ICL Aggregate In-Context Information?

With NH3, we study if ICL considers all in-context information equally. We have just seen that ICL does not treat pre-training preference equivalently to in-context label information. However, it is similarly important to understand how ICL treats different sources of purely in-context information.

To test NH3, we change the label relationship *during* in-context learning in three different scenarios. (D → F): after N observations of the **d**efault label relation we

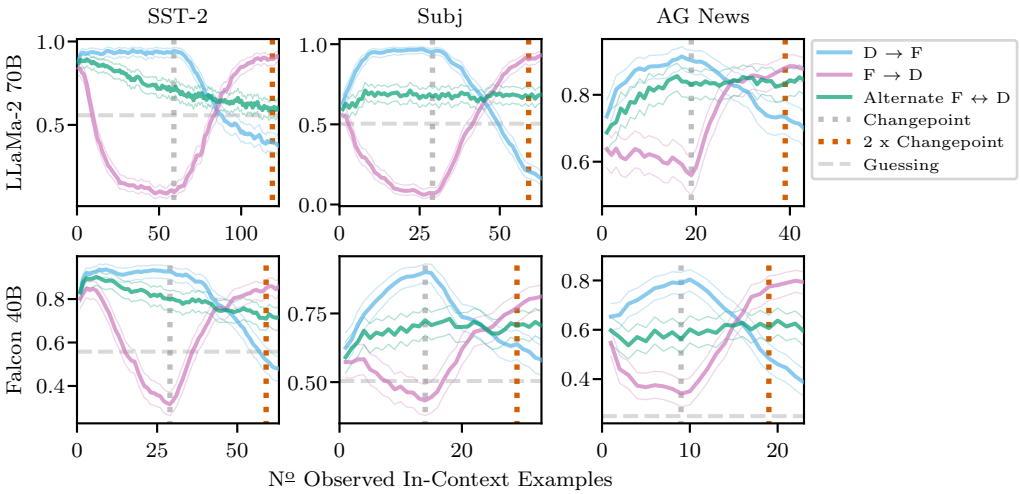


Figure 6.6: Few-shot ICL accuracies when the **label relationship changes throughout ICL**. For (D → F), we start with **d**efault labels and change to **f**lipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point – but they are not. Bootstrapped 99 % confidence intervals, moving averages (size 3), and 500 repetitions.

flip the label relation for all following observations, e.g. from (negative, positive) to (positive, negative) for SST-2. (F → D): we now start with N flipped label observations and then expose the model to default labels. (Alternate F ↔ D): we alternate between the default and flipped labels after *each* observation. For all three setups, after $2N$ observations, the model has observed the same number of the flipped and default label examples. If NH3 is true, ICL should treat all observed label relations equally, no matter their position in the input. This means, predictions should be the same for all three scenarios after $2N$ observations.

Observations & Discussion Figure 6.6 shows accuracies for a selection of models and tasks, and we report full probabilistic metrics for all tasks and model combinations for which ICL was able to learn the flipped relationships well in Figs. D.7 to D.11. We observe that, across almost all tasks and model combinations, predictions are significantly different between the three setups after observing the same number of examples of both label relationships (after $2N$ total observations,

red dashed line in the figure). ***We thus reject NH3 that ICL treats all information provided in-context equivalently.***

After the changepoint N, predictions immediately begin to adjust to the new label relationship. In particular, after $2N$ observations, the (F $\rightarrow$ D) setup has a bias for predicting according to the default label relationship, while the (D $\rightarrow$ F) setup has a bias for predicting the flipped label relationship. In other words, LLMs prefer to use information that is *closer* to the query, instead of considering all available information equally. Our finding is distinct from Zhao et al. (2021), who observe that ICL preferentially predicts labels that appear frequently near the query for a single *fixed* label relation. Lastly, we note once more that the results here also strongly support our previous rejection of NH1.

6.9 Related Work

Some recent work has studied the effect of labels in ICL. Yoo et al. (2022) also revisit label randomization and find significant variance across tasks and models. Pan et al. (2023) further separate ICL into label-independent and -dependent learning, which they study by replacing labels with arbitrary tokens. Wei et al. (2023) find that smaller or instruction-tuned models are less capable when performing ICL with replacement labels. Similar to Min et al. (2022b), the above studies do not consider probabilistic metrics or full ICL training curves, and thus can underestimate changes in ICL predictions for modified labels. For example, Pan et al. (2023) find that the gap between random and default labels is "insignificant" for small models, which our results, in particular for probabilistic metrics, contradict, cf. Section 6.5. Further, Wei et al. (2023) claim that "large models can override prior knowledge from pre-training [...] in-context" and "small models do not change their predictions when seeing flipped labels", which is not supported by our results in Section 6.7. Lastly, Gao et al. (2021) observe that replacement labels can also degrade performance when *finetuning* language models.

More generally, ICL has been the subject of many recent studies. For example, Min et al. (2022a) finetune language models to improve ICL, Si et al. (2023) measure

the inductive bias of ICL predictions, Chan et al. (2022b) and Dasgupta et al. (2022) study differences between in-weights and in-context generalization, and Chang and Jia (2022), Liu et al. (2022), and Zhang et al. (2022b) observe that the selection of examples affect ICL predictions significantly. In this chapter, we emphasize a probabilistic treatment of ICL predictions. Uncertainty in LLMs has previously been studied, e.g. by Kadavath et al. (2022), Lin et al. (2023), Bai et al. (2022a), and Gonen et al. (2023). On non-language tasks, Kirsch et al. (2022) and Chan et al. (2022a) study properties that lead to the emergence of ICL.

6.10 Discussion

Non-Parametric Prediction As ICL predicts in direct dependence on few-shot examples, we can view ICL as an instance of non-parametric prediction (cf. Section 2.1.3). However, it is a highly unusual non-parametric model. For example, ICL is often sensitive to the order of the input examples (Lu et al., 2022) – and thus does not assume that the data are IID or even exchangeable (cf. Section 2.1.2). We discuss this, as well as relations to NPTs from Chapter 3 and neural processes (Garnelo et al., 2018b) further in Chapter 7.

Alignment For alignment of LLMs, it is crucial to understand how pre-training preference and inputs trade-off, as well as how different parts of the input, such as a context string and user input, interact and influence predictions. Our results suggest prompt-based alignment (Bai et al., 2022b) may struggle to overwrite pre-training preference and could itself be overcome easily by future user input.

Do Labels Always Matter? It is plausible that labels matter less for other NLP tasks such as question answering, where in-context examples may provide limited information towards the answer of the query question. However, for the randomized label experiment, capable LLMs might still identify that the provided in-context answers are random and imitate this in their predictions.

Limitations We focus on few-shot ICL tasks where evaluation is based on logits and not free-form generation. We do this mostly to avoid complications around evaluating free-form generation tasks and believe our results should transfer to this setting. Further, our experiments do not cover RLHF-finetuned LLMs (Christiano et al., 2017; Ziegler et al., 2019; Ouyang et al., 2022).

6.11 Conclusions

In this chapter, we have investigated how the conditional label distribution of in-context examples affects ICL predictions. To ensure our conclusions represent ICL behavior well, we have studied ICL across all possible in-context dataset sizes and considered probabilistic aspects of ICL predictions. In some sense, we have shown that ICL is both better and worse than expected. On the one hand, our results demonstrate that, against expectations set by prior work, ICL does incorporate in-context label information and can even learn truly novel tasks in-context. On the other hand, we have shown that analogies between ICL and conventional learning algorithms fall short in a variety of ways. In particular, label relationships inferred from pre-training have a lasting effect that cannot be surmounted by in-context observations. Additional prompting can improve but likely not overcome this deficiency. Further, ICL does not treat all information provided in-context equally and preferentially makes use of label information that appears closer to the query.

7
Conclusions

Contents

In this last chapter, we contextualize and discuss the main contributions made in Chapters 3 to 6. We first give an overview of research that has come out since the publication of the original papers on which this thesis is based. We then explore similarities and differences between the approaches discussed in Chapters 3 to 6 and relate the discussions in this thesis to the overarching challenges in data-efficient deep learning, suggesting avenues for future work as appropriate.

7.1 Follow-up Work

At the time of writing, up to three years have passed since the individual papers on which Chapters 3 to 6 are based were first published on pre-print servers. Since then, a variety of related follow-up work has appeared that acknowledges or builds upon the contributions described in this thesis. In this section, we will give an overview of this body of related follow-up work.

Follow-up Work to NPTs A significant amount of work related to the NPT architecture of Chapter 3 has been published. Notin et al. (2023) propose the ProteinNPT model, which adapts the NPT architecture for use with protein sequence data. Most notably, motivated by their particular application setting, they change the NPT architecture to average attention maps for the ABA layer across rows, such that the column interactions are the same for each input. They show that ProteinNPT consistently outperforms baselines for protein property prediction tasks and further demonstrate application to iterative protein design.

The quadratic cost of self-attention limits the number of datapoints to which NPTs can attend simultaneously. Rastogi et al. (2023) address this by extending

the NPT architecture with cheap approximate self-attention mechanisms based on cross-attention to learned inducing points (Jaegle et al., 2021; Lee et al., 2019). This allows NPTs to scale to a larger set of input points, which Rastogi et al. (2023) demonstrate improves performance on genomic data.

Since the original publication, a number of alternative deep non-parametric approaches for tabular data have been proposed. Schäfl et al. (2022) propose Hopular, which similarly predicts in non-parametric fashion by attending to the entire dataset. Unlike NPTs, Hopular features skip connections that allow each layer to attend directly to the original input. Hollmann et al. (2023) apply attention between datapoints to meta-learn predictions on tabular data, building on their earlier work with attention-based neural processes (Müller et al., 2022). Thimonier et al. (2023) use the reconstruction loss on masked-out features of an NPT architecture for anomaly detection tasks. Wang and Sabuncu (2023) propose a deep non-parametric model that is explainable due to its simple similarity-weighted linear prediction head. Gorishniy et al. (2024) fuse nearest neighbor retrieval with deep learning, obtaining a deep non-parametric model which can effectively search over large datasets.

In the context of deep learning for tabular data, the contributions of the NPT architecture have been widely acknowledged, for example by the following publications and pre-prints: Grinsztajn et al. (2022), Borisov et al. (2022), Rubachev et al. (2022), Liao and Li (2023), Luo and Xu (2023), Aytekin (2023), Jin and Ucar (2023), and Marton et al. (2024). In particular, Rabbani et al. (2024) confirm our results that the architectural biases of ABA and ABD help prediction on tabular data: in their large scale benchmark of tabular deep learning methods, they find that SAINT (Somepalli et al., 2021) and NPTs perform best, with SAINT improving over NPTs on some datasets.

A number of works have since used architectures similar to NPTs across a variety of domains. Ke et al. (2023) use an NPT-like architecture for causal discovery. Borgeaud et al. (2022) build RETRO, a language model that predicts in non-parametric fashion by cross-attending to a set of retrieved chunks of text. Bohdal

et al. (2024) cite NPTs as inspiration for their use of cross-attention between images to solve domain adaptation tasks.

Follow-up Work on Label-Efficient Model Evaluation Since the original publications on which Chapters 4 and 5 are based, a number of works have taken an interest in label-efficient model evaluation. Pourkamali-Anaraki et al. (2023) directly apply LURE-based active testing to problems in engineering and material science. Ochiai et al. (2023) actively evaluate model accuracy by applying an active learning algorithm for level set estimation (Gotovos, 2013). Jain et al. (2023) show that sensitivity to arbitrary changes in the input correlates with model performance for LLMs, opening the door to label-free model evaluation. Yu et al. (2024) make first steps towards combining active learning with active testing with their "active testing while learning" algorithm.

Matsuura and Hara (2023), Ashury-Tahan et al. (2024), and Boubdir et al. (2023) propose methods for efficiently selecting between candidate models: while the approach of Matsuura and Hara (2023) is based on adapting LURE-based active testing, Ashury-Tahan et al. (2024) and Boubdir et al. (2023) reduce the labeling cost for choosing between text generation models by focusing labeling efforts on inputs for which candidate model predictions disagree.

Active evaluation has also been studied beyond the supervised machine learning model setting: Nayak et al. (2022) actively evaluate the quality of clustering algorithms and Li et al. (2022) design a label-efficient algorithm for two-sample tests.

A motivation of our line of work on active testing is to draw attention to the fact that labeling costs for model evaluation cannot be neglected in practical scenarios. This has since been recognized, for example, by Jesson et al. (2021), Van Amersfoort (2022), Vishwakarma et al. (2023), Wörmann et al. (2022), and Kim et al. (2023). In particular, Hutchinson et al. (2022) and Hu et al. (2018) highlight active testing in their surveys on evaluation in machine learning.

Lastly, in the context of large foundation models (Bommasani et al., 2021), recent work has highlighted the cost of model evaluation when predictions, not

labels, are expensive (Prabhu et al., 2024). For example, Polo et al. (2024) release smaller but representative versions of popular benchmarks to reduce the cost of evaluating large language models.

Follow-up Work on In-Context Learning At the time of writing, it has only been a few months since our work on in-context learning has been accepted for publication at a machine learning conference. Although the field is moving fast, this limits the amount of relevant follow-up work.

Hendel et al. (2023) propose that ICL works by iteratively accumulating knowledge about the task from in-context demonstrations in a so-called task vector, which is simply the last hidden state of the Transformer at a particular layer. They show that, when replacing the hidden state of *zero-shot* input queries with the task vector, much of the original performance of ICL can be recovered. A similar phenomenon has independently been recently reported by Todd et al. (2024).[1]

Instead of prepending examples to the input as in ICL, Liu et al. (2023b) only use the examples to derive shift vectors which are used to offset the latent activations of the Transformer. They show this can be advantageous over ICL for certain tasks, in particular when targeting the style of the generations.

Petrov et al. (2024) explore limits in the computational expressiveness of in-context learning techniques such as prompting and pre-fix tuning, finding that they fall short of full model finetuning.

Oswald et al. (2023) and Vladymyrov et al. (2024) continue to explore the idea that the capabilities of ICL can be explained by Transformers implicitly implementing gradient descent algorithms.

Similar to our label flipping experiments, Pawelczyk et al. (2023) use ICL with flipped labels to study the *unlearning* of information acquired during pre-training.

[1] If future work finds task vectors can be reliably extracted across a wide range of ICL scenarios, this suggests that non-parametric ICL might best be seen as implicitly learning a *parametric predictor*, since task vectors are fixed in dimensionality and thus have limited capacity.

7.2 Discussion and Future Work

The last chapters have presented and discussed four approaches for data-efficient deep learning. With the exception of Chapters 4 and 5, between which there is a natural connection, we have largely presented these methods as independent contributions. In this section, we will discuss similarities between the chapters as well as relate them to the larger context of the challenges currently faced in deep and efficient machine learning research. Whenever appropriate, we will also suggest interesting avenues for future work beyond those already covered in the individual chapters.

Breaking IID As hinted at throughout the thesis, probably the largest commonality between the discussed approaches for data-efficient deep learning is that they all break the assumption that the data are identically and independently distributed, or IID. We have introduced the IID assumption itself in Section 2.1.2, and we have discussed the technical reasons for why the individual approaches themselves break IID in the respective chapters. Here, we want to discuss whether or not there is a shared motivation to breaking IID. In other words, is there something about data-efficient deep learning that makes breaking IID attractive?

Before we attempt to answer this question, let us recap and discuss the motivation behind breaking IID for the individual approaches. The inspiration behind NPTs in Chapter 3 is to explore non-parametric prediction as an inductive bias for deep learning – breaking IID is a consequence of the architecture that we propose but not a core motivation. For active model evaluation in Chapters 4 and 5, however, breaking IID *is* a core part of the method: being parsimonious with our label acquisitions means we should update our acquisition function as we get more information with each iteration – which breaks IID. Lastly, for ICL in Chapter 6, breaking IID is a direct consequence of directly using the pre-trained LLM, which, as a sequence model over language, naturally, is designed and trained to be sensitive to the order in which the data arrive. Thus, ICL breaks IID as a consequence of breaking exchangeability. There is no reason to think that breaking IID is desirable or beneficial for good ICL. However, the benefits we get from accessing the highly

performant pre-trained LLM outweigh the negative that ICL does not conform to the assumption of exchangeability. Sometimes a model trained on a lot of data with bad assumptions is better than a model trained on very little data with good assumptions.

So, in summary, there seem to be a variety of reasons for breaking IID in data-efficient deep learning in this thesis. Perhaps the common denominator of these is that, when data are scarce, we need to become inventive to make deep learning data-efficient: find better architectures, be careful in our acquisition of labels, or make use of large pre-trained models. And as a consequence of this inventive spirit, sometimes deliberately but sometimes also coincidentally, the approaches discussed in this thesis end up breaking the IID assumption.

There are a variety of interesting directions for future research surrounding this discussion. For example, it may be possible to modify pre-trained LLMs such that ICL becomes exchangeable, either exactly or approximately, for example, by modifying positional embeddings or through appropriate finetuning (Min et al., 2022a). It would be interesting to see if adding exchangeability to ICL actually improves performance as common intuition would suggest.

Further, in addition to breaking IID, NPTs also defy common intuitions about overfitting. NPTs typically have millions of parameters and are explicitly designed to make fewer assumptions than established models, supporting both parametric and non-parametric prediction. Yet, they perform well on datasets with only hundreds of datapoints. Clearly, there must be a strong inductive bias in this architecture to prefer functions that generalize well. Characterizing and understanding this better is highly interesting future work.

NPTs, ICL, and Neural Processes There are a number of similarities – and some important differences – between NPTs and ICL in LLMs. For one, both rely on the Transformer architecture, although there are some differences between the exact implementations. Also, both predict in direct dependence of a set of training datapoints. However, ICL can predict for a variety of tasks, and, as we show in Chapter 6, can even generalize to new entirely tasks. In contrast, NPTs are

trained on a single task only and do not benefit from vast amounts of pre-training knowledge like ICL. And while ICL really only works well for language inputs, NPTs can easily accommodate numerical or abstract symbolical inputs.

The neural process family of architectures (Garnelo et al., 2018b), cf. Sections 2.3 and 3.3, can be used to establish a connection between ICL and NPTs. In particular, the neural process models of Kim et al. (2019), Müller et al. (2022), and Nguyen and Grover (2022) are similar to the NPT architecture. However, unlike NPTs, neural processes are trained over a collection of datasets: each input is a new draw from a prior over datasets or, equivalently, functions. This input is split into a training set, on which the neural process conditions, and a test set, for which the neural process predicts. Neural processes have been shown to closely approximate Bayesian predictive distributions (Müller et al., 2022), for example, learning to emulate Gaussian process posterior predictives. We can turn NPTs into (conditional) neural processes simply by changing the training regime such that each input is a different dataset. In fact, we have preliminary results reproducing the experiments of Müller et al. (2022) with an NPT architecture.

As for ICL, there are significant similarities to neural processes as well. Just like neural processes, ICL performs well for a variety of different input tasks or functions. It has even been shown, e.g. by Kadavath et al. (2022), that the uncertainties of LLMs typically include both an epistemic and an aleatoric contribution (Der Kiureghian and Ditlevsen, 2009; Kendall and Gal, 2017), i.e. they can represent uncertainty originating from a lack of knowledge about the task. While it is clear that ICL does not directly implement Bayesian inference as suggested by Xie et al. (2022), for example because it breaks exchangeability, comparisons between ICL and implicit Bayesian inference in neural processes should not be discarded outright. Neural processes thus allow us to relate NPTs and ICL.

Non-Parametric or Semi-Parametric Models like NPTs and ICL are sometimes referred to as semi-parametric models, see, for example, Rastogi et al. (2023) and Borgeaud et al. (2022). This highlights that these models are somewhere

between parametric and non-parametric models: unlike traditional non-parametric models, they have a large number of explicit parameters that are learned in standard parametric modeling fashion. However, *semi-parametric modeling* is already an established term with a slightly different meaning, see, for example, Ruppert et al. (2003) or Murphy (2012, Chapters 9.6.1 and 15.2.6). Semi-parametric modeling typically refers to models which mix parametric and non-parametric components, often in simple and interpretable ways, e.g. building a semi-parametric GP as $f(\boldsymbol{x}) + \boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{x})$, where f is a GP and $\boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{x})$ a linear model. This is different to models such as NPTs and ICL, which learn parametric components to realize non-parametric prediction. While this might not be ideal either, in this thesis, we have simply called these models non-parametric to highlight their predictions are in direct dependence on training data. However, perhaps an entirely new name such as "neural non-parametric models" would be the best option going forward.

In-Context Learn Everything One would expect the prevalence of in-context learning to continue to grow. With foundation models currently increasing steadily in size and capabilities, it seems possible that it will soon neither be feasible nor necessary to finetune models for most individual tasks. As such, we believe that future research into the limitations of how ICL can modify LLM predictions is important and necessary, for example, comparing ICL behavior to full model finetuning. While ICL has so far largely been limited to language, we believe that, as foundation models become inherently multimodal, ICL in other modalities, or even ICL across modalities, will become an interesting object of study. See, for example, the recent work of Balazevic et al. (2024) for a vision model capable of learning tasks in-context.

Previously, we have argued that the lack of exchangeability is a weakness of ICL. However, from the perspective of "in-context learning for everything" we arguably would not want a model that is strictly exchangeable. An exchangeable model might struggle in many practical situations where the data actually are non-stationary, such as time series data or distribution shifts in observations over

time. However, we would like a model that, when *appropriate* for the data at hand, behaves approximately exchangeable. Note that, practically speaking, we already observe that the order-sensitivity of ICL is much reduced when there are many in-context observations. It would be interesting to study more carefully the extent to which ICL already approximately respects or ignores exchangeability depending on the data at hand.

Evaluating Foundation Models The reliable evaluation of foundation models is one of the big challenges in machine learning research right now. Here, "evaluation" takes on a more holistic meaning than in Chapters 4 and 5 (Liang et al., 2023). As these models, in particular LLMs, are used to perform an ever larger number of tasks in deployed automated decision making systems – many of which may not even be known to the developer in advance – it becomes increasingly hard to make sure these models perform as intended (Celikyilmaz et al., 2020). This does not entail making sure that models can perform all possible tasks. However, we do want to make sure that they fail safely when ill-equipped for a particular task and that they further refrain from undesirable behavior such as basing decisions on harmful stereotypes.

It is impossible, both from a perspective of compute and labeling costs, to evaluate foundation models for all possible inputs they will experience at deployment. Therefore, Perez et al. (2022) and Ganguli et al. (2022) propose active and automated evaluation approaches which try to "poke holes" into LLMs to surface harmful or undesirable behavior. Unlike for active testing, the *bias* this introduces is not a particular concern here: we do not care about estimating the *average* model performance; instead, the challenge is to find as many edge cases as possible for which the model breaks.

Nevertheless, we believe that ideas from active testing, or, on a larger level, optimal experimental design (Lindley, 1956; Rainforth et al., 2024) can play an important role in designing better approaches to evaluating LLMs. For example, some ideas from active testing, such as adaptive surrogate models, diversity

in predictions between surrogate and target model, as well as good epistemic uncertainties should be helpful to guide holistic language model evaluation, too.

Integrated Efficient Deep Learning We have seen that data-efficient deep learning can take many forms, including architecture bias, transfer learning, or active acquisition. In research, these approaches are still largely presented as disparate solutions – and the chapters of this thesis are no exception to this. However, practitioners require comprehensive solutions that combine approaches to give the best possible performance in data-sparse scenarios. We would encourage more future work that considers all possible aspects of data-efficiency, instead of studying isolated components.

For example, an interesting avenue for future work is combined active learning and active testing. While first steps in this direction have been made (Yu et al., 2024), we believe additional work is necessary to improve these approaches and bring them to practitioners.[2] Our active model evaluation approaches in Chapters 4 and 5, further make only limited use of the unlabelled data, and similar to Bickford Smith et al. (2024), they could likely benefit from including more information from unlabelled examples via semi-supervised or pre-trained models.

There are a variety of directions holding back the adoption of active learning and testing methods. For example, approaches to data-efficient deep learning currently completely neglect the computational cost of model training. In practice, while the cost of labels might be higher than the cost of compute, the computation costs might not be negligible either. Ideally, we should strive to develop methods that balance data- and compute-efficiency as necessary to minimize the *total* cost of model development. Another problem with active learning or testing is that, for the practitioner, it is hard to evaluate whether or not they made the right choice going with an active methods over a standard passive approach. Large scale studies or cheap on-the-fly evaluation proxies could help develop trust in active methods.

[2]For example, Yu et al. (2024) assume a fixed and equal split between training and testing labeling budget, which cannot be optimal.

7.3 Summary

This thesis has investigated four approaches to data-efficient deep learning. With Non-Parametric Transformers in Chapter 3, we have shown that non-parametric prediction via learned self-attention is a useful inductive bias for deep learning models on tabular data. Chapters 4 and 5 have demonstrated that, when labels are expensive, we can use active testing methods to evaluate models in a label-efficient manner. Lastly, Chapter 6 has investigated the learning abilities of data-efficient in-context learning in large language models, revealing both similarities and differences compared to conventional learning algorithms. These approaches demonstrate that, perhaps counterintuitively, the continued growth of both model and dataset sizes over the last years has led data-efficient methods for deep learning, not to their demise, but into a new and exciting era.

Appendices

A

Beyond Individual Input-Output Pairs in Deep Learning

Contents

A.1 Proof – NPTs Are Equivariant over Datapoints

We here provide proof that NPT is equivariant to a permutation of the datapoints. This requires, among other things, showing that multi-head self-attention is equivariant. We were unable to find this proof in the existing literature, e.g., Set Transformer (Lee et al., 2019) relies heavily on equivariance of self-attention but does not provide proof. In the following, we will refer to datapoints as the *rows* of our input, see e.g., Fig. 3.1.

Definition 1. *A function $f : \mathcal{X}^n \to \mathcal{X}^n$ is row-equivariant if for any permutation $\sigma : [1, \dots, n] \to [1, \dots, n]$ applied to the dimensions of $\mathcal{X}^n$, we have for all i, $f(X_1, \dots, X_n)[i] = f(X_{\sigma^{-1}(1)}, \dots, X_{\sigma^{-1}(n)})[\sigma(i)]$.*

Lemma 1. *Any function of the form $f(X_1, \dots, X_n) = (g(X_1), \dots, g(X_n))$ for some g is row-equivariant. These functions are denoted as 'row-wise operations', as they consist of the same function applied to each of the rows of the input.*

Proof. Follows immediately from the structure of f. □

Lemma 2. *The composition of row-equivariant functions is row-equivariant.*

Proof. This result is widely known, but a proof here is included for completeness. Let f and g be row-equivariant.

$$f \circ g(\sigma X) = f(g(\sigma X)) = f(\sigma g(X)) = \sigma f(g(X)). \tag{A.1}$$

□

Lemma 3. *Let $W \in \mathbb{R}^{n \times m_1}$ and $X \in \mathbb{R}^{m_2 \times n}$. The function $X \mapsto XW$ is row-equivariant.*

Proof. Let σX be a permutation of the rows of X. Then we have

$$(\sigma X)W[i, j] = \sum \sigma X[i, k] W[k, j] \tag{A.2}$$

$$= \sum X[\sigma^{-1}(i), k] W[k, j] = XW[\sigma^{-1}(i), j] = \sigma(XW)[i, j]. \tag{A.3}$$

□

Lemma 4. *The function $X \mapsto Att(XW^Q, XW^K, XW^V)$ is row-equivariant.*

Proof. Let the row-wise softmax function be denoted $\omega(\cdot)$. Then we have

$$Att(XW^Q, XW^K, XW^V) = \omega(XW^Q(XW^K)^\top/\sqrt{h})XW^V, \tag{A.4}$$

where

$$\begin{aligned}
\sigma XW^Q(\sigma XW^K)^\top[i,j] &= \sigma(XW^Q)\sigma(XW^K)^\top[i,j] && \text{(A.5)}\\
&= \sum \sigma(XW^Q)[i,k]\sigma(XW^K)[j,k] && \text{(A.6)}\\
&= \sum XW^Q[\sigma^{-1}(i),k]XW^K[\sigma^{-1}(j),k] && \text{(A.7)}\\
&= XW^Q(XW^K)^\top[\sigma^{-1}(i),\sigma^{-1}(j)] && \text{(A.8)}\\
&=: A. && \text{(A.9)}
\end{aligned}$$

Note that the above result states that the function $XW^Q(XW^K)^\top$ is *not* row-equivariant because of the additional permutation of the columns. Let σ denote a permutation operator on matrices. Then straightforwardly we have the following:

$$\omega(\sigma A/\sqrt{h})) = \sigma\omega(A/\sqrt{h})\,. \tag{A.10}$$

Finally, it remains to show that the final matrix multiplication step restores the row-equivariance property we seek.

$$\begin{aligned}
\sigma \underbrace{\omega(XW^Q(XW^K)^\top/\sqrt{h})}_{=:M}(\sigma XW^V)[i,j] &= \sigma(M)(\sigma XW^V)[i,j] && \text{(A.11)}\\
&= \sigma(M)\sigma(XW^V)[i,j] && \text{(A.12)}\\
&= \sum M[\sigma^{-1}(i),\sigma^{-1}(k)](XW^V)[\sigma^{-1}(k),j] && \text{(A.13)}\\
&= M(XW^V)[\sigma^{-1}(i),j]. && \text{(A.14)}
\end{aligned}$$

Which shows that self-attention is row-equivariant. □

Lemma 5. *The following hold:*

1. *Multihead self-attention is equivariant.*
2. *If f and g are row-equivariant, then the function $x \mapsto g(x) + f(x)$ is also row-equivariant.*
3. *Res(H) is row-equivariant.*

4. *MHSA(H) is row-equivariant.*
5. *ABD is row-equivariant.*
6. *ABA is row-equivariant.*

Proof. We show each item.

1. We know that $X \mapsto O_i$ is equivariant from the previous lemma, and this trivially implies that $X \mapsto \text{concat}(O_1, \ldots, O_k)$ will also be row-equivariant. Finally, because $\sigma AB = \sigma(AB)$, get that MHSelfAtt(H) is row-equivariant.
2. Straightforward.
3. Because LayerNorm is row-equivariant (being a function applied row-wise to the matrix), Res(H) is a sum of two row-equivariant functions and so by a previous result will also be row-equivariant.
4. Because rFF is again a row-wise operation and so trivially row-equivariant, the previous results on sums and compositions of row-equivariant functions directly yield row-equivariance of MHSA.
5. ABD is by definition an application of MHSA(H), and therefore is row-equivariant by the above result.
6. ABA is a row-wise operation and is therefore trivially row-equivariant.

□

Property A.1.0.1. *NPT is row-equivariant.*

Proof. Each layer of NPT has been shown to be row-equivariant. Because NPT is a composition of such row-equivariant functions, it is therefore row-equivariant. □

A.2 Additional Experiments

A.2.1 Semi-Synthetic Experiments

Attention Maps for the Semi-Synthetic Experiments

We here display additional results for the semi-synthetic experiments of Section 3.4.2. In Fig. A.1, we display attention weights for Attention Between Datapoints (ABD) for all depths and a subset of heads of the architecture. We see that some, but not

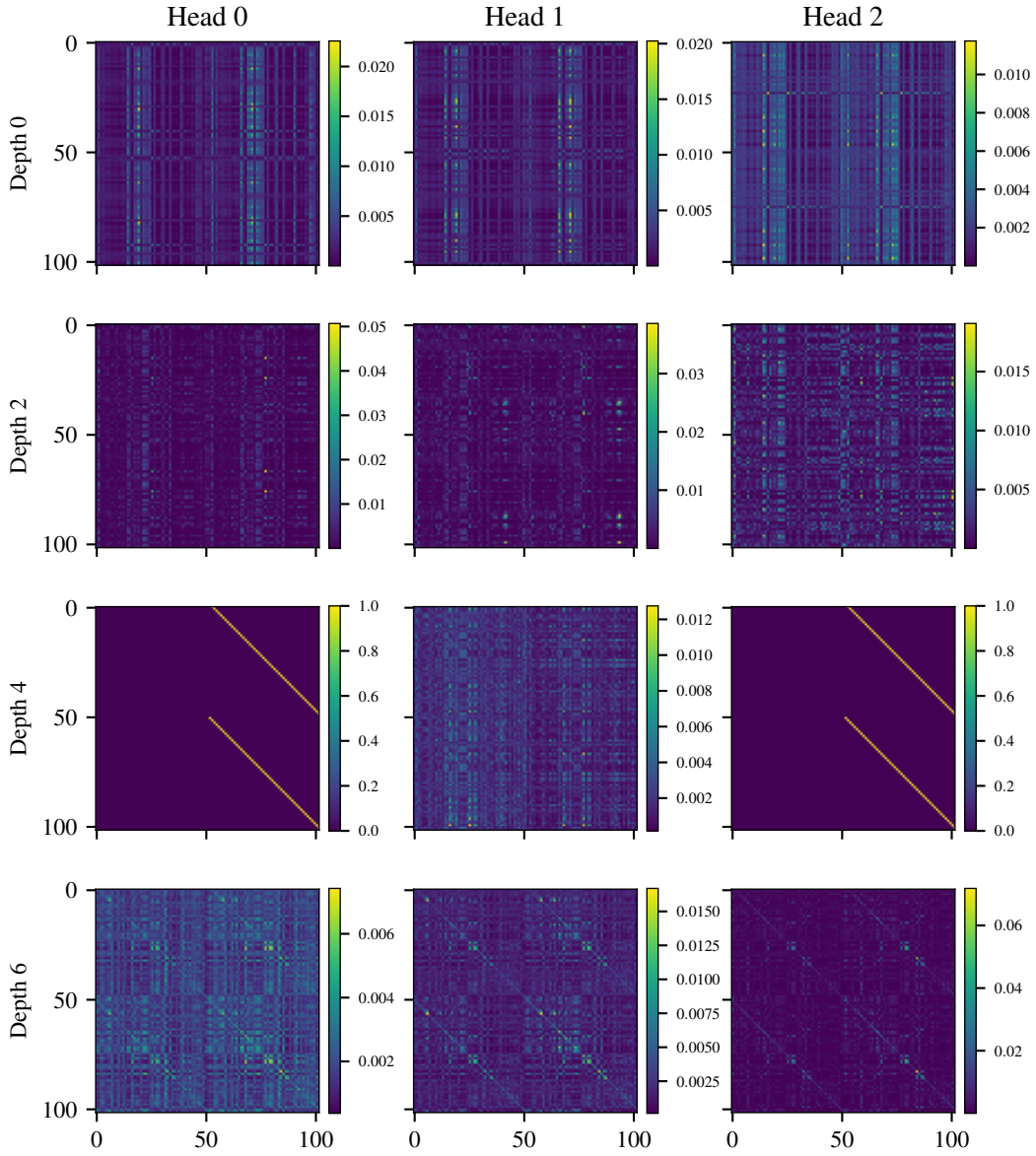


Figure A.1: Visualizations of NPT attention maps for Attention Between Datapoints (ABD) for the semi-synthetic experiment at all model depths, a selection of heads, and a single batch of input data. Evidently, not all attention maps need to perform a "lookup" for the model to solve the task. In fact, some heads appear to learn almost query-independent behavior (e.g., heads 0, 1, and 2 at depth 0).

all, attention heads display the desired diagonal lookup pattern. Note that, in this case, one head would suffice to implement lookup and perfectly solve the task.

A brief comment on the attention maps with the "double diagonal" structure (e.g., depth 4, head 0): we see that (a) original datapoints attend to the duplicate points and (b) duplicates also attend to duplicate datapoints. Behavior (a) makes sense: NPT needs to attend to the duplicates from the originals to look up the target values. This behavior in turn minimizes loss. Behavior (b) is irrelevant to loss, because NPT does not need to predict anything for the duplicates, and no loss is computed. However, (b) suggests that the query embeddings learned by the self-attention *ignore* the masked out label column in the input. Hence, the resulting queries for the originals and the duplicates would be identical – both leading to high attention values for the keys of the duplicates – and ultimately resulting in the double diagonals in Fig. A.1.

Modified Semi-Synthetic Experiments

Setup In Section 3.4.2, we mention that with some concessions the original lookup task can also be solved by standard non-parametric models. However, we also mention that simple modifications to the task make it, again, unsolvable for any model of which we are aware other than NPT. We here demonstrate these hypotheses for two non-parametric models: k-Nearest Neighbors (k-NN) and Deep Kernel Learning (DKL).

First, we apply k-NN and DKL to the original duplication tasks. As mentioned in the main text, this already requires us to make some concessions: we now need to explicitly split the input data into a global training set (all duplicated datapoints) as well as a test set (all original datapoints). That is, if all duplicate datapoints make up the training set, then non-parametric models are able to predict perfectly on the original datapoints, because most non-parametric models rely on distances in some manner, and here, distances in input feature space are sufficient to successfully match entries. This is trivially true for k-NN but also for DKL, where the RBF

kernel of the GP will lead to the desired "matching behavior" as long as the learned neural network embedding does not collapse distances.

In other words, NPTs would ideally learn a k-NN-style prediction for the semi-synthetic dataset. Crucially, while non-parametric models predict based on distances because of fixed design choices, NPTs *learn* this behavior and can just as well learn other more complicated relations between datapoints.

We now present two modifications to the semi-synthetic dataset; NPT can accommodate them because the model learns the nature of interactions, but they significantly affect the performance of the fixed kernel methods.

- **Random Features**: A subset of the features are randomized across both original and duplicate datapoints independently. Specifically, we overwrite the entries of the last three features with noise drawn independently from a Gaussian distribution $\mathcal{N}(1, 1)$. To solve the task, matches between datapoints must now be computed using the subset of non-randomized features only.
- **Add One**: We add 1 to all target regression values *only* for the duplicate datapoints. Matches can still be made based on all features, but now a 1 must be subtracted from the lookup value to solve the task.

As in the original setting, we train the models on the modified semi-synthetic datasets and check with novel test data whether they have learned the correct relational mechanism underlying the experiment.

Note that the Random Features and Add One settings also distinguish our setup from prompting in natural language processing literature (Radford et al., 2019; Brown et al., 2020) because the original datapoints are no longer "correct" input-output pairs; the model must use an underlying relational structure instead of memorization to solve the task.

Results Table A.1 presents RMSE values obtained by the models when trained on the original duplication task, the two modifications separately, as well as both modifications applied.

Table A.1: Variations of the semi-synthetic dataset that require learning of between-datapoint interactions more complex than simple lookups. While NPTs can learn complex interactions between datapoints, conventional non-parametric approaches lack flexibility and fail.

Test RMSE ↓	Original Synthetic	Random Feats.	Add One	Random Feats. + Add One
1-NN	**0.00**	7.19	6.11	7.80
k-NN	**0.00**	5.42	5.18	5.64
DKL	**0.00**	5.94	6.31	6.36
NPT	0.34	**0.24**	**0.46**	**0.75**

Evidently, for NPTs, the different scenarios do not lead to a large difference in performance; in all instances, they achieve near-perfect loss because their predictions leverage attention between datapoints. Careful optimization of NPT training convergence would likely lead to a further reduction in loss. Nevertheless, the achieved losses by NPT are more than a magnitude lower than those on the original data and correspond to a near-perfect Pearson-correlation with the target values of $r > 99.9\%$. We conclude that NPTs successfully learn to attend to the correct subset of features, to subtract 1 from the lookup target values, or to do both at the same time.

Next, we consider the non-parametric models. First, we confirm in *Original Synthetic* that the non-parametric models can indeed solve the original lookup task. However, we find that neither DKL nor k-NN can accommodate any of the modifications, reverting to an RMSE that is worse than the performance of all baselines on the original Protein dataset, see Table A.7.[1]

For k-Nearest Neighbor, $k = 1$ is clearly optimal in the original semi-synthetic setup. However, k-NN cannot learn to ignore certain attributes (Random Features) and or to modify looked-up values. Setting $k > 1$ actually improves prediction because it considers other matching points in addition to the (now misleading) duplicates for prediction. However, even with $k > 1$, k-NN does not achieve much better than guessing performance on the modified tasks.

[1]In fact, the RMSEs are about equal to the standard deviations of the target values in the Protein dataset, 6.11, such that the values obtained by the models on the modified setups amount to random guessing. We further note that we apply all modifications to the standardized input data, such that the Add One setting adds a full standard deviation for the final evaluation in Table A.1.

DKL also fails to accommodate any of the presented task modifications. We suspect that DKL, in theory, should be able to solve the Random Features task. That is, DKL should be able to use the neural network to learn a representation that discards any information from the randomized columns. We were unable to achieve this, but it may be possible with additional adaptations to the model. Ideally, we would condition the GP on new "test data" (the duplicates) in each mini-batch during training. This was not easily possible with the GPyTorch codebase (Gardner et al., 2018). At test time however, we did directly reconstruct an exact GP using embedded inputs and RBF scale parameters learned during training.

In any case, DKL can never solve the Add One scenario because, after independently transforming features with a neural network, DKL simply applies a GP in embedding space. This means that it will always naively interpolate target values between training data (duplicates) and test data (features) in embedding space, and cannot *learn* interactions between points, such as subtracting 1 from all duplicate targets.

Even further, there is another easy option of how to construct this experiment such that only NPT will be able to solve it: we could *randomly sample the attribute* for which we mask out the entry, i.e., all columns can now be target columns. All non-parametric models presented here rely on a fixed set of features as input to predict for a fixed target column. They are not compatible with this style of "imputation" problem, i.e., there is no way to even take as input data like this in such models. NPTs, however, take both features and targets as input, only using the masking mechanism to distinguish between features and targets as well as train and test data. Hence, they can easily adapt to this scenario.

The bad results for the non-parametric models also highlight that these models must predict non-parametrically, unlike NPT, which could always fall back to parametric prediction if it cannot learn the interactions required for a task.

(k)-NN Hyperparameter details We use the scikit-learn (Pedregosa et al., 2011) implementation of (k)-Nearest Neighbors, where we exhaustively search for neighbors by setting `algorithm=brute` and otherwise use default parameters. For 1-NN, we set $k = 1$, for k-NN we sweep over $k \in [1, \ldots, 10]$ and report results for the k that achieved the best performance.

DKL Hyperparameter details We use the GPyTorch implementation of Deep Kernel Learning. We perform a non-exhaustive random sweep over a selection of hyperparameters and select those with best validation performance. This results in the following changes from the default hyperparameter values: for the Original Synthetic and Add One scenario we disable dropout, use hidden layers $[100, 100]$, a learning rate of 0.0001, train for a maximum of 30000 epochs, with 256 inducing points, 8 features, batch size of 128, and early stopping patience on the validation loss of 20 epochs. For the Random Features and the Random Features + Add One scenarios, we arrive at the same configuration, except that we train with 64 inducing points.

A.2.2 Attention Between Datapoints on Real Data

Corruption Experiments

In our Data Corruption experiments in Section 3.4.3, we make use of Algorithm 3 below. When predicting for a datapoint k, this algorithm completely destroys information from all other datapoints $i \neq k$ in the batch b by randomly permuting attribute values across all other datapoints. Therefore, if NPT's loss increases after corruption, it must meaningfully rely on attention between datapoints for prediction.

Algorithm 3: Data Corruption

Input: list of masked mini-batches $\mathcal{B} = [\boldsymbol{X}^{(b)} \in \mathbb{R}^{K \times d} \mid b \in 1 \dots B]$, unmasked label column $\boldsymbol{X}_{:,d}$, trained model $f : \boldsymbol{X}^{(b)} \to \boldsymbol{X}^{(b)}$, batch size K, loss function $\mathcal{L}$, number of attributes (including features and target) d

Returns: test loss under data corruption $\mathcal{L}^{\text{corr}}$

$\mathcal{L}^{\text{corr}} \leftarrow 0$
for $\boldsymbol{X}^{(b)}$ *in* $\mathcal{B}$ **do**
  for k *in* $1 \dots K$ **do**
    $\boldsymbol{X}^{(b,k)} \leftarrow \boldsymbol{X}^{(b)}$ `// initialize batch to be corrupted`
    for j *in* $1 \dots d$ **do**
      $\boldsymbol{X}^{(b,k)}_{i \neq k,j} \leftarrow \texttt{permute}_{\text{axis}=i}(\boldsymbol{X}^{(b,k)}_{i \neq k,j})$ `// permute each attr. column indep.`
    end
    $\mathcal{L}^{\text{corr}} \mathrel{+}= \mathcal{L}(f(\boldsymbol{X}^{(b,k)})_{k,d}, \boldsymbol{X}_{k,d})$ `// compute loss w/ unmasked label column`
  end
end
return $\mathcal{L}^{\text{corr}}$

Alternatively, we could also input datapoints *individually*, i.e., decrease the mini-batch size to 1, to test if NPT depends on attention between datapoints. Indeed, we find that performance also deteriorates in this scenario. However, we believe that the Data Corruption experiment provides stronger evidence because it preserves batch statistics across attributes. This makes sure that performance deterioration is not caused by spurious factors, such as a decreased batch size that was not encountered in training. While NPT is generally compatible with varying batch sizes, we leave a thorough investigation of this for future work.

A.2.3 Real Data – To Which Other Points Does NPT Attend?

Attention Maps on Real Data

In Fig. A.2, we display ABD attention maps of NPT for the Protein regression dataset in addition to the one shown in Section 3.4.4. For visualization purposes, we sort the input datapoints with respect to their feature space distance to an arbitrary test datapoint. This is to ensure that the global structure of the attention maps in Fig. A.2 has meaning. Specifically, nearby entries in the attention maps belong

to input datapoints that are close in input space. With this transformation, the diagonal patterns appearing in Fig. A.2 clearly suggest that our model is attending more strongly between datapoints that are similar in input space. Similar to the semi-synthetic experiments, some but not all attention heads display this pattern of interest.

Data Deletion Experiment

We here give full details on the Data Deletion experiment presented in Section 3.4.4. To recap, we consider the prediction of NPT for a single test sample i^*. We then iteratively delete other datapoints from the input if they do not significantly change the prediction of NPT on i^*. Algorithm 4 describes this in detail. We are then interested in differences between the deleted and the kept datapoints. Specifically, we compare the average feature space distance in input space between the active datapoint i^* and either the kept datapoints $\mathcal{R}$ or deleted datapoints $\{1, \dots, n\} \setminus (\{i^*\} \cup \mathcal{R})$, obtaining average distances $D_{i^*,\text{kept}}$, $D_{i^*,\text{deleted}}$. We break out of the deletion algorithm if less than $\epsilon\%$ of the original points remain, to reduce variance in our estimates of the kept statistic. We repeat Algorithm 4 for all 5567 test points $i^* \in \mathcal{D}_{\text{test}}$ in the Protein regression dataset.

We perform a Wilcoxon signed-rank test on the pairs $\{D_{i^*,\text{kept}}, D_{i^*,\text{deleted}}\}_{i^* \in \mathcal{D}_{\text{test}}}$ to determine if the median of the kept datapoints is less than the median of the deleted ones. The test is highly significant at $p \approx 0$, i.e., smaller than the floating point precision of SciPy Stats allows. The raw Wilcoxon statistic is 3125889.5.

To make sure the difference is not an effect of sample size, we also construct a set of average differences to a set of randomly drawn datapoints.[2] That is, instead of using Algorithm 4 for *targeted* deletion, we *randomly* construct $\mathcal{R}$, essentially only applying lines 10 and 15 of Algorithm 4. For each active test row i^*, we randomly

[2]There are many fewer kept than deleted datapoints. Further, there are outliers in the dataset, and these affect the deleted datapoints more often than the kept datapoints. We find that the average distance between a *random* subset and the *deleted* (not the kept!) datapoints also becomes statistically significantly smaller at large sample sizes. Hence, we compare the *deleted* datapoints to a *random* subset to control for size effects.

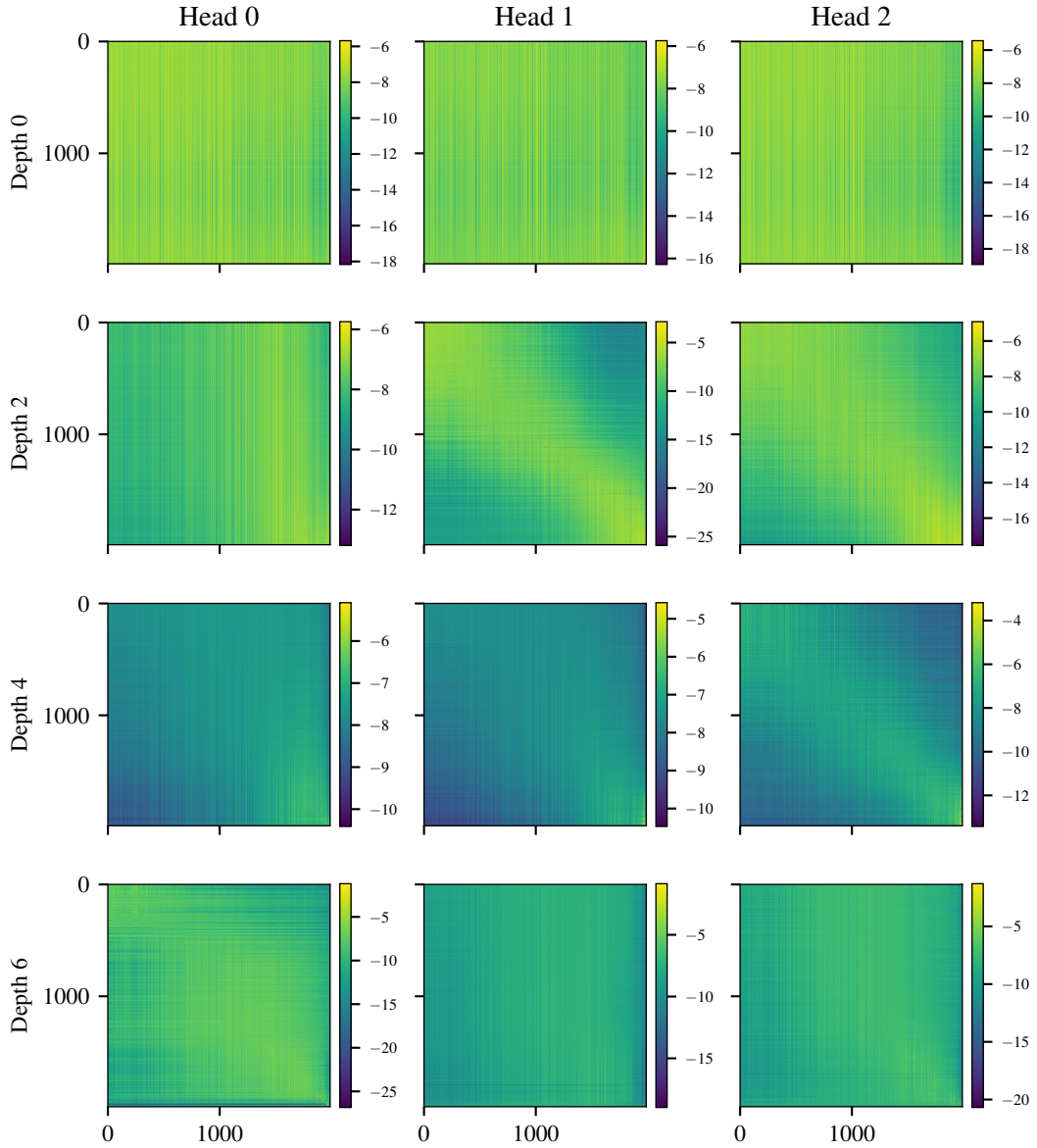


Figure A.2: Visualizations of the Attention Between Datapoints (ABD) attention maps for real data – here, the Protein regression dataset – for all depths and a selection of heads. Input to the model is sorted such that datapoints that are similar in input space have nearby indices. The diagonal pattern (e.g., depth 2 and head 1) indicates that the model attends to similar inputs more strongly. For illustration purposes, we here plot the log of the attention values.

Algorithm 4: Data Deletion

Input: Masked data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, active sample index i^*.

$\hat{y} \leftarrow \mathrm{NPT}(\boldsymbol{X})_{i^*,d}$ `// original NPT prediction at active datapoint`

$\Delta_{\max} \leftarrow 0.1$ `// maximum allowed change in prediction`

$\Delta_{\mathrm{it}} \leftarrow 0.01$ `// initialize maximum change per deleted datapoint`

$N_{\text{max-retry}} \leftarrow 50$ `// maximum number of retries before increasing` $\Delta_{\mathtt{it}}$

$\epsilon \leftarrow 0.02$ `// fraction of points remaining at which we break`

$\mathcal{R} \leftarrow \{1, \ldots, n\} \setminus \{i^*\}$ `// initialize remaining set`

$N_{\mathrm{retry}} \leftarrow 0$ `// initialize no. of retries`

while *True* **do**

  $c =$`random_choice`(R) `// random proposal for data deletion`

  $\hat{y}_{\mathrm{proposal}} = \mathrm{NPT}(\boldsymbol{X}_{(\mathcal{R}\setminus\{c\})\cup\{i^*\}})_{i^*,d}$ `// predict without proposed datapoint`

  $\Delta_{\mathrm{proposal}} = \frac{|\hat{y}_{\mathrm{proposal}} - \hat{y}|}{\hat{y}}$ `// change in pred. when deleting proposal`

  if $\Delta_{\mathrm{proposal}} < \Delta_{it}$ **then**

    if $\Delta_{\mathrm{proposal}} < \Delta_{max}$ **then**

      $\mathcal{R} \leftarrow \mathcal{R} \setminus \{c\}$ `// delete datapoint from input`

      $N_{\mathrm{retry}} \leftarrow 0$

    else

      break `// exceeded maximum change`

  else

    $N_{\mathrm{retry}} \leftarrow N_{\mathrm{retry}} + 1$ `// candidate change was too large, try again`

  if $N_{\mathrm{retry}} \geq N_{\text{max-retry}}$ **then**

    $\Delta_{\mathrm{it}} \leftarrow 1.1 \cdot \Delta_{\mathrm{it}}$ `// increase allowed change per iteration`

    $N_{\mathrm{retry}} \leftarrow 0$

  if $|\mathcal{R}| < \epsilon \cdot n$ **then**

    break `// less than` $\epsilon\%$ `of original datapoints remaining`

end

return $\mathcal{R}$

delete as many datapoints as were deleted in targeted fashion. A Wilcoxon signed-rank test between the distances for the random and kept subset is likewise significant at $p \approx 8.77 \cdot 10^{-130}$. This is the value we report in the main body.

We also run a computationally more demanding version of the algorithm with $\Delta_{\mathrm{it}} \leftarrow 0.005$, $\epsilon \leftarrow 0.01$ to see how many points we can successfully delete. This

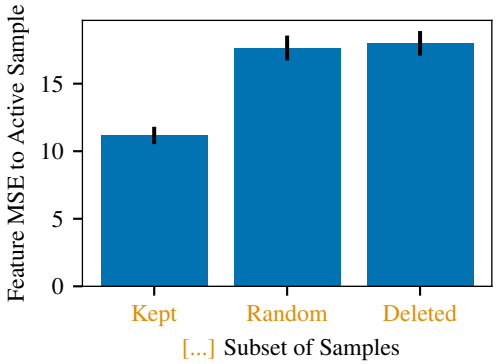


Figure A.3: When predicting for any given datapoint, NPT prefers to keep similar datapoints around. Displayed are average feature space differences and their standard errors between the active datapoint and the sets of kept, random, and deleted datapoints for a single batch.

version of the algorithm requires more computation which is why we limit execution to the test datapoints of a single batch. The results are statistically significant at $5.26 \cdot 10^{-49}$ for kept $<$ deleted and $8.38 \cdot 10^{-39}$ for kept $<$ random for a Wilcoxon signed-rank test. We illustrate the differences between the distances in Fig. A.3. We further note that using Algorithm 4, we are able to reduce the set of datapoints present in the input to 1% of the original n for 79.5% of active test datapoints and to 10% in 99.5% of cases. Percentages refer to $n = 2048$ datapoints in total, of which 398 were test datapoints.

All in all, these experiments strongly suggest that NPT relies on interactions between similar datapoints for prediction.

A.2.4 Ablation Study 1: NPT Hyperparameters

We conduct an ablation study on the Protein and Boston Housing datasets (Table A.2). For Protein, the same 0.7/0.1/0.2 train/validation/test split is used for all model configurations. Boston Housing uses a 0.7/0.2/0.1 train/validation/test split with 10-fold cross-validation.

Despite the significant difference in dataset sizes between Boston Housing ($n = 506$) and Protein ($n = 45730$), and the fact that Boston Housing includes both categorical and continuous variables, the base models used for each dataset are nearly identical.

Table A.2: NPT ablation study: test root mean-squared error (RMSE) on the Protein and Boston Housing regression datasets.

Test RMSE ($\pm$ Std Err) $\downarrow$	Protein	Boston
Base NPT	3.41	3.00 ± 0.23
No Semi-Supervision	3.38	3.38 ± 0.46
No Target Masking	3.32	2.93 ± 0.18
No Feature Masking	3.56	2.95 ± 0.21
No Feature Masking, No Target Masking	3.58	3.20 ± 0.26
Feature Mask $p = 0.15 \rightarrow p = 0.5$	3.87	3.39 ± 0.23
Target Mask $p = 0.15 \rightarrow p = 0.5$	3.37	3.11 ± 0.28
8.00 $\rightarrow$ 4.00 Layers	3.43	3.30 ± 0.41
8.00 $\rightarrow$ 16.0 Layers	3.36	3.05 ± 0.24
8.00 $\rightarrow$ 4.00 Heads	3.42	3.25 ± 0.30
8.00 $\rightarrow$ 16.0 Heads	3.37	3.20 ± 0.39
Tradeoff λ $= 0.5$	3.50	2.96 ± 0.25

On both datasets, we use an NPT model with 8 layers, 8 heads, per-attribute hidden dimension $e = 128$, feature and target masking with $p = 0.15$ for each, a cosine annealing schedule for the loss tradeoff λ, the LAMB (You et al., 2020) optimizer with Lookahead (Zhang et al., 2019), a flat-then-anneal learning rate schedule with cosine decay and base learning rate 0.001, dropout with rate 0.1 on the attention weights and after linear layers, and gradient clipping at 1. This configuration is essentially the same as the `NPT-Base` configuration described in Appendix A.3.1, which we use with minimal per-dataset modifications for all other results in this work.

The Boston Housing model takes as input the full dataset (i.e., batch size = 507) and Protein uses mini-batching with batch size = 2048. Boston Housing trains for 20 000 steps, and Protein for 400 000. The learning rate is constant for the first 70 % of steps for Protein, but only for the first 50 % of steps for Boston, starting the learning rate annealing earlier to defend against overfitting on the small dataset. These changes directly result from the different dataset sizes.

As Table A.2 shows, the performance of NPT is robust to a variety of significant hyperparameter choices. This illustrates that practitioners will likely *not need to spend much time tuning hyperparameters* when applying NPT to novel datasets. We

now give results for the ablation study on the Protein and Boston datasets separately.

Protein Dataset See Table A.2 for results and performed ablations. It is computationally too expensive for us to perform full cross-validation over all ablations for the Protein regression dataset. Instead, we report the results of a single 5-fold cross-validation for the Base NPT configuration on Protein (also varying the model random state). This results in an RMSE of 3.40 ± 0.05 (σ). The standard deviation of the 5-fold cross-validation allows us to roughly gauge which ablations have significant effect. Given the results in Table A.2, we find that the majority of ablations do not lead to meaningful changes in performance. Only the somewhat dramatic changes to the optimization of NPT result in its performance falling from the top rank on the Protein Dataset (second rank CatBoost has RMSE = 3.51): removing stochastic feature masking ($p_{\text{feature}} = 0$), removing both stochastic feature masking ($p_{\text{feature}} = 0$) and stochastic target masking ($p_{\text{target}} = 1$, training targets are always masked out at training time and NPT therefore cannot learn to attend to training targets at test time), or changing p_{feature} to 0.5 (meaning that 50% of all input features are masked out). NPT appears to be particularly robust to changes in model complexity, e.g., depth and number of heads, although the results suggest that we could have further increased the size of Base NPT to achieve slightly higher performance.

Boston Dataset See Table A.2 for results and performed ablations. For the Boston dataset, we repeat ablations over all 10 CV splits. Similarly, ablations on the Boston dataset are largely inconsequential; none of them result in a statistically significant change in performance from the base model. The second rank performer on Boston is MLP, at RMSE = 3.32. Only ablation of semi-supervision or changing p_{feature} to 0.5 result in a change in the top ranking of NPT among the baselines.

Altogether, the ablation study supports the claim that NPT can be applied successfully with very little tuning to datasets of vastly different sizes and feature types. Changes in model depth and number of heads do not appear significant, but

using a reasonably low feature masking probability (e.g., 15%, as has been commonly used in the literature (Devlin et al., 2019)) may be important to stable training.

Supported by these ablations, we sweep over only a small selection of configurations for our main benchmark comparison in Section 3.4.1. And indeed, it seems that NPT is robust to hyperparameter changes, given that these configurations perform well across vastly different settings (binary and multi-class classification, datasets with millions of datapoints, etc.) than those explored in the ablations. See Appendix A.4 for details.

We speculate that NPT's robustness stems from (a) being a relatively over-parametrized architecture that is powerful enough to model a wide variety of datasets and (b) from the effective regularization introduced by the feature masking mechanism. Finally, we emphasize that the aim of this work is to introduce the NPT architecture and examine its properties, not to spend significant effort and compute resources on achieving top performance across all benchmarks.

A.2.5 Ablation Study 2: NPT without ABA and NPT without Feature Masking

We next present an additional ablation study targeting two core components of NPTs across all datasets: the Attention Between Attributes (ABA) layer and the stochastic feature masking.

ABA Layer First, we perform an ablation to test if ABA layers are beneficial in practice. For this, we simply leave out the ABA layers, such that the MLP at the end of the ABD layers (see "rFF" in Eq. (3.6)) is now the only way for the model to independently transform the features of input datapoints.

Our results, given in Table A.3, show that, generally, ABA is a useful component of the NPT architecture. Leaving out ABA increases performance only for 3/10 datasets. Interestingly, all three of these datasets are regression tasks, which may warrant further investigation. We observe the largest difference for the Poker Hands dataset, which requires complex reasoning between input features: in the same number of training steps, the ablation only achieves 57.4% accuracy compared to

Table A.3: Additional ablation studies. We study ablations of NPT (a) without ABA layers and (b) without stochastic feature masking. In both cases, performance tends to decrease. These results suggest that both ABA layers and stochastic feature masking contribute positively to the performance of NPTs. For the small datasets, we report mean values and standard errors over 10 CV splits.

	NPT without ABA	NPT without Feature Masking	Default NPT
Classification			
Poker Hand (Acc. ↑)	57.4	69.7	**99.3**
Forest Cover (Acc. ↑)	95.5	96.0	**96.7**
Higgs Boson (AUC ↑)	0.859	0.871	**0.892**
Income (AUC ↑)	**0.952**	**0.952**	**0.952**
Kick (AUC ↑)	0.767	0.766	**0.770**
Breast Cancer (AUC ↑)	0.992 ± 0.008	0.996 ± 0.006	$\mathbf{0.997 \pm 0.001}$
Regression			
Boston Housing (RMSE ↓)	3.22 ± 0.25	3.18 ± 0.35	$\mathbf{2.92 \pm 0.15}$
Yacht (RMSE ↓)	1.15 ± 0.11	$\mathbf{0.50 \pm 0.06}$	1.27 ± 0.15
Concrete (RMSE ↓)	$\mathbf{4.79 \pm 0.12}$	5.37 ± 0.20	5.21 ± 0.20
Protein (RMSE ↓)	**3.29**	3.59	3.41

99.3% for full NPT. These results support our hypothesis that ABA is useful when the dataset requires complex transformations of the features. Our most general recommendation would be to default to using NPTs with ABA layers, as they boost performance on the majority of datasets we examine. However, if practitioners can spend the extra compute, exploring NPTs without ABA can be worthwhile.

Stochastic Feature Masking We perform an ablation to test if the stochastic feature masking objective (cf. §3.2.6) is beneficial in practice. For this, we simply disable all stochastic masking of input features by setting $p_{\text{features}} = 0$.

Our results, also in Table A.3, show that for 9/10 datasets, enabling feature masking yields at least a small improvement in performance. Disabling feature masking is detrimental to the performance on the Poker Hands dataset, leading to a 30% drop in accuracy. Again, our general recommendation would be to use NPTs with feature masking by default, as it rarely seems to decrease performance and sometimes helps significantly, but to explore NPTs without feature masking if feasible.

Table A.4: UCI classification datasets: test accuracy. Standard error reported for datasets with multiple cross-validation splits.

Test Accuracy ↑	Higgs Boson	Poker Hand	Forest Cover	Income	Kick	Breast Cancer
Random Forest	76.2	71.5	94.8	95.4	90.1	94.20 ± 0.70
Gradient Boosting	76.5	94.1	96.7	95.8	90.2	94.03 ± 0.90
XGBoost	77.0	95.9	**97.1**	95.6	**90.3**	94.91 ± 0.68
CatBoost	76.6	99.2	95.7	**95.8**	90.1	$\mathbf{95.61 \pm 0.75}$
LightGBM	75.9	92.8	85.0	**95.8**	**90.3**	95.26 ± 0.82
MLP	78.3	**99.5**	95.2	95.4	90.0	94.73 ± 0.89
k-NN[3]	—	50.4	90.7	94.8	87.7	95.26 ± 0.79
TabNet[4]	77.1	53.3	94.2	95.5	89.5	94.91 ± 0.76
NPT	**80.7**	99.3	96.7	95.6	90.0	94.73 ± 0.69

Table A.5: UCI classification datasets: negative log-likelihood (NLL). Standard error reported for datasets with multiple cross-validation splits.

Test NLL ↓	Higgs Boson	Poker Hand	Forest Cover	Income	Kick	Breast Cancer
Random Forest	0.489	0.843	0.191	0.126	0.305	0.142 ± 0.012
Gradient Boosting	0.477	0.379	0.109	0.111	0.296	0.185 ± 0.024
XGBoost	0.471	0.178	**0.080**	0.147	**0.293**	0.143 ± 0.025
CatBoost	0.476	0.065	0.120	**0.109**	0.296	$\mathbf{0.124 \pm 0.024}$
LightGBM	0.486	0.420	0.361	**0.109**	0.294	0.163 ± 0.034
MLP	0.452	**0.028**	0.131	0.118	0.333	0.545 ± 0.254
k-NN[5]	—	0.975	0.274	0.139	0.333	0.466 ± 0.167
TabNet	0.469	0.973	0.151	0.119	0.314	0.233 ± 0.036
NPT	**0.412**	0.119	0.087	0.115	0.299	0.137 ± 0.026

A.2.6 Extended Results for Tabular Data Benchmarks

See Table A.4 (Table A.5) for test accuracies (negative log-likelihood scores) on the UCI classification datasets and additionally Table A.6 for AUROC results on the binary classification datasets. For the regression datasets, see Table A.7 for RMSE scores and Table A.8 for MSE scores.

[3] Out-of-memory on the Higgs Boson dataset when attempting approximate 3-NN on an Azure D64 v3 instance with 256 GB RAM.

[4] TabNet had notably lower accuracy in our setup on the Poker Hand dataset (which has a fixed test set) than that the 99.2% reported in the original work (Arik and Pfister, 2021). Our results on Higgs Boson match the reported performance more closely (78.44% (theirs) vs 77.1% (ours)). Further, we note that our other baselines achieve significantly better performance on the same datasets than those reported in (Arik and Pfister, 2021); e.g., our MLP achieves 99.5% accuracy on Poker Hand dataset while they report 50.0%; our XGBoost achieves 97.1% on Forest Cover while they report 89.34%. However, we note that some of the datasets – such as Forest Cover – do not have fixed test sets. Therefore, we cannot exclude the possibility that the performance differences are due to differently chosen train-test splits.

[5] See above note on out-of-memory.

Table A.6: UCI classification datasets: test area under the receiver operating characteristic curve (AUROC) on binary classification tasks. Standard error reported for datasets with multiple cross-validation splits.

Test AUROC ↑	Higgs Boson	Income	Kick	Breast Cancer
Random Forest	0.847	0.947	0.759	0.989 ± 0.003
Gradient Boosting	0.850	0.955	0.769	0.987 ± 0.004
XGBoost	0.854	0.946	0.775	0.989 ± 0.003
CatBoost	0.851	**0.956**	0.773	0.992 ± 0.003
LightGBM	0.843	**0.956**	**0.776**	0.992 ± 0.003
MLP	0.867	0.949	0.739	0.982 ± 0.007
k-NN[6]	—	0.932	0.747	0.980 ± 0.005
TabNet	0.857	0.948	0.745	0.978 ± 0.005
NPT	**0.892**	0.952	0.770	$\mathbf{0.997 \pm 0.001}$

Table A.7: UCI regression datasets: test root mean-squared error (RMSE). Standard error reported for datasets with multiple cross-validation splits.

Test RMSE ↓	Protein	Concrete	Boston Housing	Yacht
Random Forest	3.57	5.48 ± 0.18	3.78 ± 0.33	0.91 ± 0.13
Gradient Boosting	3.61	4.70 ± 0.18	3.44 ± 0.22	$\mathbf{0.85 \pm 0.12}$
XGBoost	3.60	4.68 ± 0.15	3.39 ± 0.29	0.88 ± 0.13
CatBoost	3.51	$\mathbf{4.28 \pm 0.16}$	3.44 ± 0.34	1.05 ± 0.16
LightGBM	3.65	4.64 ± 0.18	3.86 ± 0.27	13.60 ± 0.73
MLP	3.62	5.53 ± 0.20	3.32 ± 0.39	0.91 ± 0.13
k-NN	3.77	8.51 ± 0.30	4.27 ± 0.37	12.02 ± 0.65
TabNet	3.59	5.85 ± 0.15	3.88 ± 0.34	3.41 ± 1.12
NPT	**3.41**	5.21 ± 0.20	$\mathbf{2.92 \pm 0.15}$	1.27 ± 0.15

[6] See above note on out-of-memory.

Table A.8: UCI regression datasets: test mean-squared error (MSE). Standard deviation reported for datasets with multiple cross-validation splits.

Test MSE (± Std Dev) ↓	Protein	Concrete	Boston	Yacht
Random Forest	12.8	30.4 ± 6.4	15.4 ± 9.5	0.986 ± 0.818
Gradient Boosting	13.0	22.4 ± 5.2	12.3 ± 4.9	$\mathbf{0.867 \pm 0.779}$
XGBoost	13.0	22.1 ± 4.2	12.3 ± 7.6	0.939 ± 0.881
CatBoost	12.3	$\mathbf{18.6 \pm 4.3}$	13.0 ± 9.8	1.36 ± 1.12
LightGBM	13.3	21.9 ± 5.3	15.6 ± 7.6	190.0 ± 65.1
MLP	13.1	31.0 ± 6.9	12.6 ± 11.0	0.994 ± 0.937
k-NN	14.2	73.3 ± 16.0	19.6 ± 11.0	149.0 ± 52.6
TabNet	12.9	34.4 ± 5.8	16.2 ± 11.0	24.1 ± 54.3
NPT	**11.6**	27.6 ± 7.6	$\mathbf{8.77 \pm 2.60}$	1.80 ± 1.49

A.3 Additional Details on the NPT Architecture

A.3.1 NPT Training and Hyperparameters

NPT-Base Architecture

Below, we outline the `NPT-Base` model configuration. The final configurations used for each dataset are essentially the same as `NPT-Base`, with minor alterations in parameters such as hidden dimension size, learning rate warmup, batch size, and number of training steps. Given our limited memory and compute time budget, these changes directly result from differences in number of datapoints/attributes between the datasets. We divide the `NPT-Base` configuration into architectural details and optimization details.

`NPT-Base` Architecture

- 8 layers, alternating Attention Between Datapoints and Attention Between Attributes.
- 8 heads.
- Row-wise feed-forward (rFF) networks with one hidden layer, 4x expansion factor, and GeLU activation (standard in Transformer literature (Vaswani et al., 2017; Narang et al., 2021)).
- Attention weight and hidden layer dropout with $p = 0.1$ (cf. Appendix A.3.2).
- Per-attribute hidden dimension $e = 64$.

`NPT-Base` Optimization

- LAMB (You et al., 2020) optimizer with $\beta = (0.9, 0.999)$ and $\epsilon = 1e-6$, and a Lookahead (Zhang et al., 2019) wrapper with slow update rate $\alpha = 0.5$ and $k = 6$ steps between updates.
- Stochastic feature masking probability $p_{\text{feature}} = 0.15$.
- Anneal the tradeoff λ between feature and target loss with a cosine schedule, starting at 1 (all feature loss) to 0 (all target loss) over the course of training.

- Flat-then-anneal learning rate schedule: flat at the base learning rate for 70% of steps, and then anneals following a cosine schedule to 0 by the end of training.
- Base learning rate 1e-3.
- Gradient clipping at 1.

On all datasets with mini-batching, we approximately maintain relative train, validation, and test datapoint proportions in each batch. We train NPT in semi-supervised mode (cf. Appendix A.3.4) but have found that this does not consistently improve performance compared to conventional training because the amount of unlabeled test data is usually comparatively small.

NPT Training on Small Data

Here we describe the hyperparameter sweep details for small datasets – Breast Cancer, Boston, Concrete, and Yacht.

Base Hyperparameter Configurations Across these small datasets, we make a few minor adjustments to the `NPT-Base` architecture and optimization to obtain the `NPT-Small` configuration: we increase the default number of hidden dimensions to $e = 128$, fix the flat-then-anneal schedule to be flat for 50% instead of 70% of steps, and train with the entire dataset as input, i.e., no mini-batching. We set stochastic target masking probability to $p_{\text{target}} = 1$ by default, i.e., deterministically mask out train labels as would be done in a normal supervised setting, and then introduce modifications in our sweep.

Note that the vast majority of hyperparameters such as the number of layers and heads, optimizer, p_{feature}, tradeoff annealing schedule, learning rate schedule, and gradient clipping are exactly the same between `NPT-Base` and `NPT-Small`.

We would like to keep the base configuration for each of the small datasets exactly the same. However, we need to slightly vary the learning rate and number of epochs per dataset to optimize loss convergence across datasets. We use a base learning rate 5e-4 on Breast Cancer and 1e-3 on the other small datasets. We train for 2000 epochs on Breast Cancer and Boston, and 10 000 epochs on Yacht

and Concrete. On Breast Cancer, we additionally drop $e = 32$ due to memory constraints (it has more attributes than other small datasets).

Small Data Sweep Based on these configurations, we sweep over the following 8 configurations of the model on each dataset.[7]

- Vanilla `NPT-Small` model for given dataset.
- Increase number of layers $8 \rightarrow 16$.
- Increase number of heads $8 \rightarrow 16$.
- Increase number of layers $8 \rightarrow 16$, and number of heads $8 \rightarrow 16$.
- Stochastic target masking with probability $p_{\text{target}} = 0.1$.
- Stochastic target masking with probability $p_{\text{target}} = 0.5$.
- Increase stochastic feature masking probability from 0.15 to $p_{\text{feature}} = 0.2$.
- Use a cosine cyclic learning rate scheduler with two cycles, initial learning rate 1e-7, final learning rate 1e-7, and max learning rate given by the base model learning rate.

For the stochastic target masking variants, we proportionally increase the number of epochs (e.g., with $p_{\text{target}} = 0.5$, half as many targets are observed in a given epoch, so we double the total number of epochs).

Small Data Variant Rank Orders We report the rank order ($\pm$ standard error) of these variants in Table A.9. A notable trend is that the *target masking configurations perform particularly well.* One of the two configurations with target masking is the top performer on each of the four datasets. This could be attributed to some combination of the representational advantage of label masking (cf. Section 3.2.6), an additional regularization effect akin to dropout, or stabler convergence over a greater number of epochs.

Other configurations did not display similarly obvious trends in performance. This is in concordance with the ablation study (Appendix A.2.4) and supports the claim that NPT is robust to changes in hyperparameters.

[7] Note that we do not search a 2^8 grid over these modifications. We only try out these 8 distinct models.

Table A.9: Average rank order of variants of `NPT-Small` (± standard error) across 10 cross-validation splits on each small dataset. We determine rank using negative log-likelihood and sort methods by ascending rank for each metric.

Dataset	Boston
$p_{\text{target}} = 0.5$	2.50 ± 0.73
$p_{\text{target}} = 0.1$	2.50 ± 0.83
$8 \rightarrow$ 16 Layers, $8 \rightarrow$ 16 Heads	2.60 ± 0.65
Cosine Cyclic LR Schedule	3.10 ± 0.75
Base `NPT-Small`	3.70 ± 0.84
$8 \rightarrow$ 16 Layers	4.30 ± 0.67
$8 \rightarrow$ 16 Heads	4.40 ± 0.60
$p_{\text{feature}} = 0.2$	4.90 ± 0.46

Dataset	Breast Cancer
$p_{\text{target}} = 0.1$	2.60 ± 0.92
Base `NPT-Small`	2.70 ± 0.65
$8 \rightarrow$ 16 Heads	3.00 ± 0.49
$p_{\text{feature}} = 0.2$	3.20 ± 0.68
$p_{\text{target}} = 0.5$	3.50 ± 0.56
Cosine Cyclic LR Schedule	4.10 ± 0.89
$8 \rightarrow$ 16 Layers, $8 \rightarrow$ 16 Heads	4.40 ± 0.70
$8 \rightarrow$ 16 Layers	4.50 ± 0.81

Dataset	Concrete
$p_{\text{target}} = 0.5$	2.30 ± 0.76
Cosine Cyclic LR Schedule	2.50 ± 0.69
$8 \rightarrow 16$ Heads	2.60 ± 0.62
Base `NPT-Small`	2.70 ± 0.52
$p_{\text{target}} = 0.1$	3.10 ± 0.64
$8 \rightarrow 16$ Layers	3.90 ± 0.80
$8 \rightarrow 16$ Layers, $8 \rightarrow$ 16 Heads	5.10 ± 0.66
$p_{\text{feature}} = 0.2$	5.80 ± 0.39

Dataset	Yacht
$p_{\text{target}} = 0.1$	1.20 ± 0.53
$p_{\text{target}} = 0.5$	2.70 ± 0.52
$8 \rightarrow$ 16 Heads	2.80 ± 0.66
Cosine Cyclic LR Schedule	3.10 ± 0.69
Base `NPT-Small`	3.60 ± 0.54
$8 \rightarrow 16$ Layers	4.10 ± 0.74
$p_{\text{feature}} = 0.2$	5.20 ± 0.47
$8 \rightarrow 16$ Layers, $8 \rightarrow$ 16 Heads	5.30 ± 0.83

NPT Training on Medium and Large Data

For the medium and large datasets, we again adopt the `NPT-Base` architecture and optimization hyperparameters, and make minor manual changes on a per-dataset basis to account for differences in number of datapoints and attributes across the datasets. No more than 3 manual iterations are performed to find these adaptations. We generally attempt to maximize batch size given a fixed memory budget. Given the rank order results on small data (cf. Table A.9) we use target masking on the medium and large datasets whenever computationally feasible.[8]. These per-dataset alterations are reported below.

UCI Datasets We report results for Protein using the Base NPT configuration in the ablation study (cf. Table A.2). On Kick, we use batch size 4096, train for 250 000 steps, and use $p_{\text{target}} = 0.5$. On Income, we use batch size 2048, train for 2 000 000 steps, use no feature masking (and correspondingly fix the tradeoff parameter $\lambda = 0$), and use $p_{\text{target}} = 0.15$. On Poker Hand, we use batch size 4096, train for 200 000

[8]Training is slower as only a p_{target} proportion of training labels are used for backpropagation in each epoch. Therefore, target masking may increase training time beyond our budget.

steps, use $p_{\text{target}} = 0.5$, and stratify by class (i.e., compose training datapoints in each mini-batch proportionally to the empirical label distribution of the training set to account for significant class imbalance). On Forest Cover, we use batch size 1800, train for 800 000 steps, use a polynomial decay learning rate scheduler with warmup over the first 1% of steps, use base learning rate 0.005, $p_{\text{target}} = 0.5$, and class balancing as above. The changes to learning rate scheduling were made to speed up training and hence save compute resources. On Higgs, we use batch size 4096, train for 500 000 steps, and do not use target masking due to constraints on compute time.

A.3.2 Further Details on ABD and ABA Layers

Dropout

In practice, we apply elementwise dropout on the attention scores $\exp(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{h})$, as well as on the input/output embeddings and the output of the MHSelfAtt$(\cdot)$ function (often referred to as attention and hidden dropout).

A.3.3 Input and Output Embeddings

Input Embedding

At a high-level, we embed inputs by encoding categorical attributes as one-hot vectors and standardizing continuous attributes, followed by a learned linear embedding for each attribute to obtain $\text{InputEmbed}(\boldsymbol{X}) = \boldsymbol{H}^{(0)} \in \mathbb{R}^{n\times d\times e}$.

More specifically, we perform the following sequence of steps: Attributes $\boldsymbol{X}_{:,j}, j \in \{1,\dots,d\}$ of the input matrix can be either continuous or categorical. We first apply a function Encode$(\cdot)$ to each attribute $\boldsymbol{X}_{:,j}$. This "encodes" categorical attributes with a one-hot representation and standardizes continuous attributes to zero mean and unit standard deviation. Each encoded attribute j has (potentially unique) dimensions $n \times e_j$. Then, we concatenate this encoded attribute with its respective column of the masking matrix $\boldsymbol{M}_{:,j}$ along the second dimension to produce a column encoding of dimensions $n \times (e_j + 1)$. We learn separate embedding weights for each attribute $\boldsymbol{W}_j^{\text{in}} \in \mathbb{R}^{(e_j+1)\times e}$ that embed all attributes to a common hidden dimension

e. Altogether, we can state the embedding of a single attribute column $\boldsymbol{X}_{:,j}$ as

$$\boldsymbol{H}^{(0)}_{:,j} = \underset{\text{axis}=e}{\text{concat}}(\text{Encode}(\boldsymbol{X}_{:,j}), \boldsymbol{M}_{:,j})\boldsymbol{W}^{\text{in}}_j + \boldsymbol{H}^{\text{Index}}_{:,j} + \boldsymbol{H}^{\text{Type}}_{:,j}, \tag{A.15}$$

where $\boldsymbol{H}^{\text{Index}}_{:,j} \in \mathbb{R}^{n\times e}$ is a learned embedding for the index and $\boldsymbol{H}^{\text{Type}}_{:,j} \in \mathbb{R}^{n\times e}$ for the type (either continuous or categorical) of attribute j.

Finally, we write the full NPT input embedding layer as

$$\text{InputEmbed}(\boldsymbol{X}) = \underset{\text{axis}=d}{\text{stack}}(\boldsymbol{H}^{(0)}_{:,1}, \ldots, \boldsymbol{H}^{(0)}_{:,d}) = \boldsymbol{H}^{(0)} \in \mathbb{R}^{n\times d\times e}. \tag{A.16}$$

The stack operation constructs $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n\times d\times e}$ from d attribute embeddings $\boldsymbol{H}^{(0)}_{:,j} \in \mathbb{R}^{n\times e}, j \in \{1, \ldots d\}$.

Output Embedding

For an NPT with L layers, we obtain an output prediction by applying a learned linear output embedding (that closely mirrors the process of the input embedding) to the output of the last attention layer $\boldsymbol{H}^{(L)}$. We write the output embedding layer as

$$\text{OutputEmbed}(\boldsymbol{H}^{(L)}) = [\boldsymbol{Z}_{:,1}, \ldots, \boldsymbol{Z}_{:,d}] = \boldsymbol{Z}, \tag{A.17}$$

$$\text{where } \boldsymbol{Z}_{:,j} = \boldsymbol{H}^{(L)}_{:,j,:}\boldsymbol{W}^{\text{out}}_j. \tag{A.18}$$

Our prediction $\boldsymbol{Z}$ is a list of d attribute predictions $\boldsymbol{Z}_j \in \mathbb{R}^{n\times e_j}$. We learn output embedding weights $\boldsymbol{W}^{\text{out}}_j \in \mathbb{R}^{e\times e_j}$ which are applied on attribute slices $\boldsymbol{H}^{(L)}_{:,j,:} \in \mathbb{R}^{n\times e}$ of the output of the Lth layer $\boldsymbol{H}^{(L)} \in \mathbb{R}^{n\times d\times e}$. Note that the second dimension of each attribute prediction $\boldsymbol{Z}_j$ is determined by the encoding size (i.e., $e_j = 1$ for continuous attributes, e_j is the number of categories for a categorical attribute) as in the input embedding. Note also that we do not predict a mask value (i.e., we do not predict to dimensions $n \times (e_j + 1)$ for each attribute). To obtain the final prediction matrix $\hat{\boldsymbol{X}} \in \mathbb{R}^{n\times d}$ we take the arg max over the categorical predictions.

A.3.4 NPT Masking

Handling Missing Values

Real-world data – particularly tabular data – often contains *missing entries*. Many popular models for supervised prediction on tabular data cannot accommodate missing values as input. Instead they require that missing features are *imputed*, i.e., an additional model predicts a surrogate value for what the missing values could have been, such that the supervised model then receives a "clean" dataset as input which no longer overtly contains missing values.

For example, all scikit-learn (Pedregosa et al., 2011) predictors, including Gradient Boosting and Random Forests, require an explicit imputation step before training. Often, extremely simple imputation methods are used in practice. For example, TabNet (Arik and Pfister, 2021) drops datapoints with >10% missing entries and otherwise applies univariate mean imputation as part of a Google AI Platform pipeline (Platform, 2021); and CatBoost (Prokhorenkova et al., 2018) treats a missing continuous entry as the minimum or maximum of that feature (univariate min/max imputation), or raises an error. While more complex imputation methods could in theory be applied as pre-processing (Buuren and Groothuis-Oudshoorn, 2011; King et al., 2001; Honaker and King, 2010; Stekhoven and Buehlmann, 2012; Su et al., 2012), there will always remain a separation between the imputation step and the prediction model. Additionally, more complex imputation methods often require training and hyperparameter selection, such that the combined imputation and prediction process becomes cumbersome. Both for practical as well as performance reasons, it is desirable to have a single model that can *directly* handle missing data, learn complex internal imputation operations from the data, and at the same time learn the desired predictive function from features to target.

This is exactly what NPTs achieve. They are able to accommodate inputs with missing values gracefully without requiring any imputation pre-processing steps, therefore modeling data with missing values end-to-end. We can explicitly indicate that a value $\boldsymbol{X}_{i,j}$ is missing by simply setting the mask token $\boldsymbol{M}_{i,j} = 1$. Already in standard NPTs, the stochastic feature masking during training teaches NPTs to

predict values for which $\boldsymbol{M}_{i,j} = 1$ while ignoring the value of their entry $\boldsymbol{X}_{i,j}$ at input. Further, no choice of fixed imputation algorithm has to be made with NPTs. Instead, NPTs learn directly from the data how to make predictions given missing values. Attention between datapoints might be particularly useful for learning a general mechanism of how to impute missing values by attending to other datapoints. We therefore suspect that NPTs could be a strong contender for predicting on data with missing values. Further, unlike common imputation pre-processing, NPTs do not discard the information of *which* attributes were missing. Future work could also explore the ability of NPT to model arbitrary correlations underlying the pattern of which data is missing, i.e., datasets where values are not missing at random.

Masking Encompasses Many Common Machine Learning Settings

The flexible masking mechanism of NPTs can be used to accommodate a variety of common machine learning settings.

Multi-Target Prediction In *multi*-target classification or regression, more than one column of the dataset contains targets. Standard supervised models often do not support multi-output settings and must resort to training multiple models, one for each target. NPTs can accommodate multi-target prediction trivially, since they learn to make predictions at any masked input entry. For prediction in a multi-target setting, we simply apply target masking on all columns with targets.

Self-Supervision In self-supervised learning, we are often interested in learning a generative model or useful encoding from unlabeled data. The reconstruction of corrupted input features as part of stochastic feature masking can already be seen as self-supervised learning. The stochastic masking mechanism allows NPTs to learn to predict masked out values anywhere in the input. In theory, NPTs should be able to learn a fully generative model of the dataset in this manner.

Semi-Supervision In semi-supervised learning, we hope to use large quantities of unlabeled data to aid in learning a predictive function on a small set of labeled data. Often, this involves a two-step process, such as learning a powerful autoencoder from all data and then training a predictor using the learned encoder and the small set of labeled data. NPTs can accommodate semi-supervised learning without changes to the architecture. Specifically, we can include large amounts of unlabeled data by simply appending those feature values to the labeled input dataset. We indicate that no labels are available for all unlabeled datapoints i' by setting their mask token at the target column $\boldsymbol{X}_{i',d} = 1$. NPTs can use attention between datapoints to make use of information from the features of the unlabeled datapoints.

Imputation With imputation, we refer to scenarios where the main task is to predict missing values for arbitrary attributes and datapoints. Similar to self-supervision, NPTs already learn how to do this from the stochastic masking mechanism that is enabled by default. (Unlike for the self-supervision category, the imputation scenario assumes that there are actually some missing values that we would like to predict.)

Stochastic Masking: Details

For stochastic masking, a specified proportion of training entries (we default to 15% following (Devlin et al., 2019)) are selected for masking at the start of each epoch. Among those entries chosen, we mask out the value with 90% probability and randomize it with 10% probability. "Masking out" means that the original value $\boldsymbol{X}_{i,j}$ is overwritten with zeros and the mask token is set to 1. Randomization is done for categorical targets by sampling a new class uniformly at random. Continuous targets are sampled from a standard Normal $\mathcal{N}(0, 1)$.

This sampling scheme is applied for both stochastic feature masking and stochastic target masking, where we allow for different masking proportions between the two (p_{feature} and p_{target}). During training, a loss is backpropagated for predictions at locations with masked input entries.

A.3.5 NPT Optimization

Each of the losses $\mathcal{L}^{\text{Features}}$ (feature loss) and $\mathcal{L}^{\text{Targets}}$ (target loss) is normalized by the number of entries on which it is evaluated.

As described in Appendix A.3.1: we anneal the λ parameter in the NPT objective using a cosine schedule, i.e., starting with full weight on the feature loss term at epoch 0 and annealing to full weight on the target loss term by the end of training. We use LAMB (You et al., 2020) with Lookahead (Zhang et al., 2019) for optimization, which we find to perform well with large mini-batches. We use a flat-then-anneal learning rate schedule with cosine decay, notable as Transformer works (Vaswani et al., 2017; Devlin et al., 2019) often report that a linear learning rate warmup is necessary for training stability. Our placement of Layer Normalization before self-attention – "pre-LayerNorm" (Ba et al., 2016; Chen et al., 2018) – may contribute to our not needing this.

A.4 Classification and Regression Benchmark Details

A.4.1 General Setup

For certain datasets we use a canonical fixed test set. Otherwise, we default to 10-fold cross validation with 0.7/0.2/0.1 splits on smaller datasets and a single 0.7/0.1/0.2 split on larger datasets, where the exact split indices are always consistent across baselines. The full details on all UCI benchmark datasets are given in Tables A.10 and A.11. Note the variety of the datasets across number of instances, number of features, composition (categorical or continuous) of features, and task (multi-class classification, binary classification, and regression).

A.4.2 Hyperparameter Tuning

Overview

Table A.12 lists the number of unique hyperparameter configurations swept over for each baseline and classification/regression dataset.

Table A.10: UCI classification dataset statistics and experimental setup details.

Dataset	Higgs Boson	Poker Hand	Forest Cover	Income	Kick	Breast Cancer
# Instances	11,000,000	1,025,010	581,012	299,285	72,983	569
# Features	28	10	54	42	32	31
# Categorical Features	0	10	44	36	18	0
# Continuous Features	28	0	10	6	14	31
# Classes	2	10	7	2	2	2
Train/Val/Test Split	0.84/0.12/0.05	0.017/0.003/0.98	0.7/0.1/0.2	0.57/0.1/0.33	0.7/0.1/0.2	0.7/0.2/0.1
Fixed Test Set	Yes	Yes	No	Yes	No	No (10-Fold CV)
Uses Mini-batching	Yes	Yes	Yes	Yes	Yes	No

Table A.11: UCI regression dataset statistics and experimental setup details.

Dataset	Protein	Concrete	Boston	Yacht
# Instances	45,730	1030	506	308
# Features	9	9	13	6
# Categorical Features	0	0	2	5
# Continuous Features	9	9	11	1
Train/Val/Test Split	0.7/0.1/0.2	0.7/0.2/0.1	0.7/0.2/0.1	0.7/0.2/0.1
Fixed Test Set	No	No (10-Fold CV)	No (10-Fold CV)	No (10-Fold CV)
Uses Mini-batching	Yes	No	No	No

Table A.12: Number of unique hyperparameter configurations swept over for each model class and dataset. Here we shorten Boston Housing to BH, Breast Cancer to BC, Poker Hand to PH, Forest Cover to FC, and Higgs Boson to HB. Datasets are ordered by increasing number of datapoints (n) from left to right.
* TabNet on Protein, Kick, and Income is tuned by sweeping over all 6 configurations listed in the original paper (Arik and Pfister, 2021) in addition to the default configuration. Note that these configs include one tuned on Income.
† TabNet on Poker Hand, Forest Cover, and Higgs Boson use precisely the configuration specified for those datasets in the original paper (Arik and Pfister, 2021).
‡ For some of these, we manually optimized convergence of the validation loss by adjusting non-architectural parameters such as learning rate (schedule), batch size, or number of steps in at most 3 iterations. See Appendix A.3.1.

Dataset	Yacht	BH	BC	Concrete	Protein	Kick	Income	PH	FC	HB
Random Forest	24	24	24	24	24	24	24	24	24	24
Gradient Boosting	48	48	48	48	48	48	48	48	48	48
XGBoost	48	48	48	48	48	48	48	48	48	48
CatBoost	48	48	48	48	48	48	48	48	48	48
LightGBM	48	48	48	48	48	48	48	48	48	48
MLP	11,340	11,340	11,340	11,340	270	270	270	270	270	6
k-NN	480	480	480	480	40	40	40	40	40	-
TabNet	48	48	48	48	7*	7*	7*	1†	1†	1†
NPT	8	8	8	8	1‡	1‡	1‡	1‡	1‡	1‡

All details on the NPT hyperparameter setup are given in Appendix A.3.1. Note that for any given dataset, NPT is tuned over fewer configurations than the baselines: we fix a base model configuration with minimal data-dependent tuning of hyperparameters such as learning rate, scheduler, number of steps, and target masking percentage p_{feature}, and choose the largest batch size viable for our hardware. On small datasets, we then sweep over 8 variants, and on medium and large datasets (including image data) use only the fixed variant with minor modifications.

In the case of TabNet, the configurations used for Poker Hand, Forest Cover, and Higgs Boson are those reported by the original authors for these datasets (Arik and Pfister, 2021); for Income, we performed a sweep over configurations including one reported for that dataset in the original publication. All deep learning approaches (MLP, TabNet, and NPT) use early stopping on the validation target loss.

Baseline Sweep Details

We report hyperparameter sweep details for baselines below. The associated tables for each baseline give the bounds of the search space for numerical hyperparameters and all values for categorical hyperparameters. We clarify specific hyperparameters and provide context where helpful.

Random Forest (Tables A.13, A.14) `criterion` refers to the split criterion. `max_features` is the number of features to consider when looking for the best split.

Gradient Boosting, XGBoost, LightGBM, and CatBoost (Table A.15)

MLP (Tables A.16, A.17, A.18) The invscaling `learning_rate` scheduler scales with $\alpha_t = \alpha_0/t^{0.5}$ where t is the step, α_0 the initial learning rate, and α_t the learning rate at step t. The adaptive `learning_rate` divides the current learning rate by 5 when two consecutive epochs fail to decrease training or validation log loss by a tolerance 1e-4. Due to compute constraints, we decreased the size of the search space as the dataset size increased by focusing on 3-layer networks, lower L2 penalties, and higher batch sizes.

k-NN (Tables A.19, A.20, A.21) `weights` describes the weight function applied to the neighborhood, i.e., "distance" means that closer neighbors of a query point have greater influence than those further away. `algorithm` specifies the underlying k-NN algorithm, where KD Tree (Bentley, 1975) and Ball Tree (Liu et al., 2006) are approximations of brute-force search. The "auto" setting determines an appropriate algorithm based on the input data (Pedregosa et al., 2011). `leaf_size` is a hyperparameter of KD Tree and Ball Tree. `p` is the power parameter for the distance metric, i.e., $\texttt{p} = 1$ yields Manhattan and $\texttt{p} = 2$ Euclidean distance. It was computationally infeasible for us to obtain reasonable results on the 11M instance Higgs Boson dataset. Even when attempting approximate 3-NN on an Azure D64 v3 instance with 256 GB RAM, we encountered an out-of-memory error.

Table A.13: Random Forest classification hyperparameters.

Hyperparameter	`criterion`	`n_estimators`	`max_features`
Setting	gini, entropy	[50, 1000]	auto, sqrt, log2

Table A.14: Random Forest regression hyperparameters.

Hyperparameter	`criterion`	`n_estimators`	`max_features`
Setting	mae, mse	[50, 1000]	auto, sqrt, log2

Table A.15: Gradient Boosting, XGBoost, LightGBM, and CatBoost hyperparameters (for both regression and classification).

Hyperparameter	`learning_rate`	`max_depth`	`n_estimators`
Setting	[1e-3, 0.3]	[3, 10]	[50, 1000]

Table A.16: MLP hyperparameters for small datasets (Boston Housing, Breast Cancer, Concrete, and Yacht).

Hyperparameter	`hidden_layer_sizes`	`l2_penalty`
Setting	[(25)-(500), (25,25)-(500,500), (25,25,25)-(500,500,500)]	[0, 1]

Hyperparameter	`batch_size`	`learning_rate`	`learning_rate_init`
Setting	[32, 256]	constant, invscaling, adaptive	[1e-5, 1e-1]

Table A.17: MLP hyperparameters for medium and large datasets other than Higgs Boson (Protein, Kick, Income, Poker Hand, Forest Cover).

Hyperparameter	`hidden_layer_sizes`	`l2_penalty`
Setting	[(25,25,25)-(500,500,500)]	[0, 1e-2]

Hyperparameter	`batch_size`	`learning_rate`	`learning_rate_init`
Setting	[128, 256]	constant, invscaling, adaptive	[1e-5, 1e-1]

Table A.18: MLP hyperparameters for the Higgs Boson dataset.

Hyperparameter	`hidden_layer_sizes`	`l2_penalty`
Setting	(500,500,500)	0

Hyperparameter	`batch_size`	`learning_rate`	`learning_rate_init`
Setting	[512, 1024]	constant	[1e-4, 1e-2]

Table A.19: k-NN hyperparameters for small datasets (Boston Housing, Breast Cancer, Concrete, and Yacht).

Hyperparameter	`n_neighbors`	`weights`	`algorithm`	`leaf_size`	`p`
Setting	[2, 100]	uniform, distance	ball_tree, kd_tree, brute	[10, 100]	1, 2

Table A.20: k-NN hyperparameters for medium-large datasets (Protein, Kick, Income, Poker Hand).

Hyperparameter	`n_neighbors`	`weights`	`algorithm`	`leaf_size`	`p`
Setting	[2, 1000]	distance	auto	[10, 100]	2

Table A.21: k-NN hyperparameters for Forest Cover.

Hyperparameter	`n_neighbors`	`weights`	`algorithm`	`leaf_size`	`p`
Setting	[2, 25]	distance	auto	[10, 100]	2

B

Active Testing: Label-Efficient Model Evaluation

Contents

B.1 Additional Experiments

B.1.1 Synthetic Data

Figure B.1 shows experiments on additional synthetic data together with the experiments familiar from Fig. 4.3 of the main chapter. We show two additional regression settings: a GP on sinusoidal data with non-uniform densities in x and a GP on the quadratic data already familiar from Fig. 4.3 (b). Active testing yields gains in sample-efficiency for both of these additional settings. We give details for all synthetic experiments in Appendix B.3.2.

B.1.2 Radial BNN on MNIST

We also investigate active testing of a Radial BNN on MNIST in a large data regime with 50 000 training points. As acquisition strategy, we here simply use the predictive entropy of the Radial BNN, which we observe to be well-calibrated in this simple setting. Consequently, we see large gains in sample-efficiency of active testing over naive IID acquisition for this setting as shown in Fig. B.2. We discuss experimental details for this plot in Appendix B.3.8

B.1.3 Uncertainties and Statistical Significance

In Figs. B.3 to B.5, we show a variation of Figures 4.4, 4.6 (a), and 4.8 (b) wherein we plot *mean* log squared differences instead of *median* squared differences.[1] All conclusions from the main chapter continue to hold for the mean-based visualization. Additionally, we quantify uncertainties on the mean values via the standard errors of the log squared differences.

We also investigate the statistical significance of the results reported in the chapter, computing Wilcoxon signed-rank test statistics to examine if the best active testing strategy has lower population mean rank than all other shown methods. More precisely, when comparing two methods at a fixed acquisition step, we obtain a pair of samples of squared differences for each run, and we test against the

[1] Reminder: We calculate the squared difference between the true test loss on the full test set (known only to an oracle) and the estimator at each acquisition step.

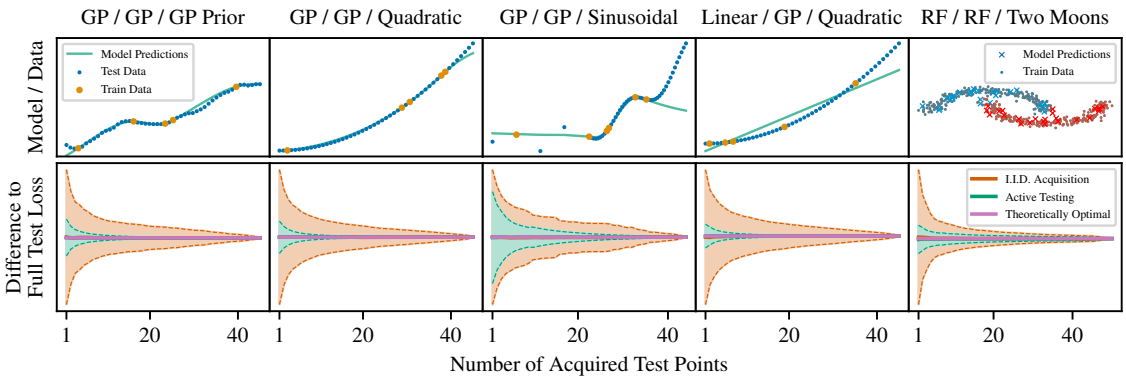


Figure B.1: Active testing on synthetic data. Columns (1, 4, 5) are repeated from Fig. 4.3 of the main chapter, while (2–3) are exclusive to the appendix. Each column shows a different combination of *model/surrogate/data*. The first row shows model predictions, as well as the points used for training and testing for a single draw from the random data generation. The second row displays the mean difference of the estimators to the true loss on the full test set (known only to an oracle).

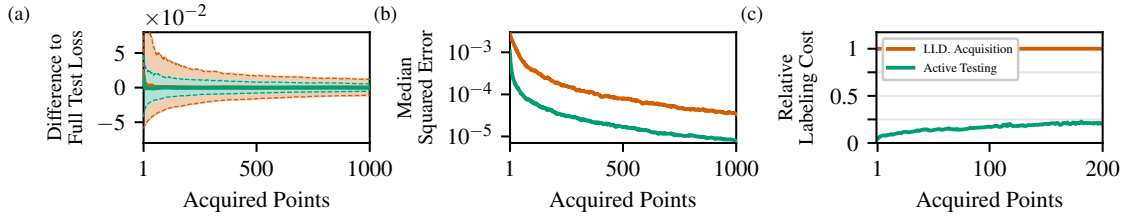


Figure B.2: Active testing for a Radial BNN on **MNIST** using model predictive entropy to estimate test cross-entropy loss. (a) Active testing gives unbiased loss estimators with (b) much lower median squared error. (c) We calculate the *relative labelling cost*: the sample-efficiency factor of IID acquisition to active testing at any number of acquired points. Lower is better and values less than 1 are gains in sample-efficiency. We plot medians and quantiles $(0.1, 0.9)$ and average over 992 runs.

alternative hypothesis that the best performing method has *lower* squared error. We compare methods at the last displayed acquisition step. The results of the test show that we can always reject the null hypothesis at 5×10^{-3} confidence level. We give the full results of the Wilcoxon signed-rank tests in Table B.1.

Table B.1: Wilcoxon signed-rank tests for experiments from the main chapter, comparing the squared differences of the best method against all other shown approaches. The alternative hypothesis is that the "best method" has lower squared error than the "other method".

Figure	Dataset	Best Method	Other Method	Acq. Step	p-Value
4 a	MNIST	BNN Surrogate	IID Acquisition	1000	5.438×10^{-61}
	MNIST	BNN Surrogate	Random Forrest Surrogate	1000	7.069×10^{-19}
	MNIST	BNN Surrogate	Original Model	1000	9.524×10^{-20}
4 b	Fashion-MNIST	Random Forrest Surrogate	Original Model	1000	6.586×10^{-125}
	Fashion-MNIST	Random Forrest Surrogate	IID Acquisition	1000	4.176×10^{-31}
	Fashion-MNIST	Random Forrest Surrogate	ResNet Surrogate	1000	4.249×10^{-17}
	Fashion-MNIST	Random Forrest Surrogate	ResNet Train Ensemble	1000	4.254×10^{-10}
6 a	CIFAR-100	ResNet Train Ensemble	IID Acquisition	500	3.010×10^{-11}
	CIFAR-10	ResNet Train Ensemble	IID Acquisition	500	3.988×10^{-76}
	Fashion-MNIST	ResNet Train Ensemble	IID Acquisition	500	2.457×10^{-105}
7	Fashion-MNIST	Predictive Entropy	IID Acquisition	400	2.801×10^{-6}
	Fashion-MNIST	Predictive Entropy	Mutual Information	400	8.779×10^{-4}

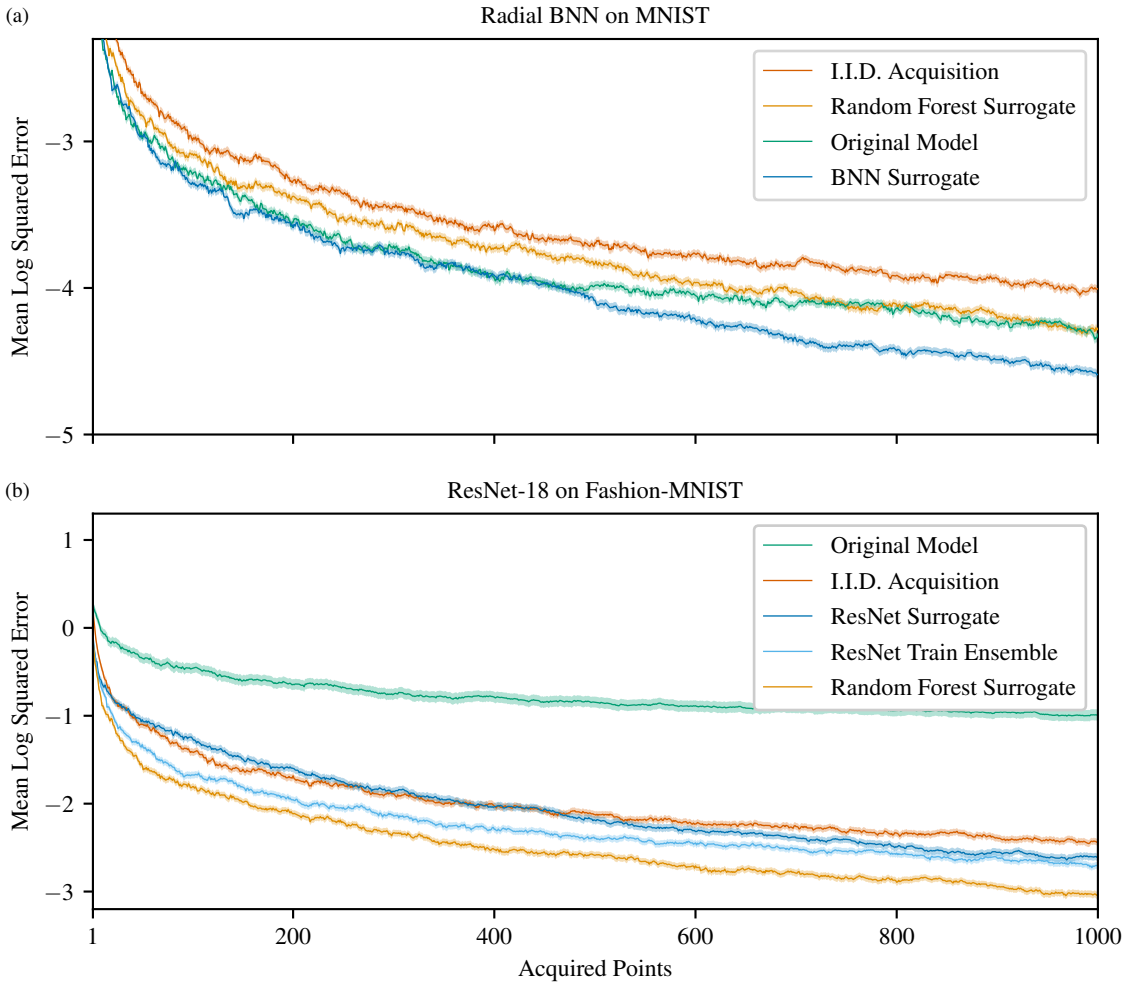


Figure B.3: Fig. 4.4 from the main chapter but now showing the mean of the log squared difference instead of the median. Additionally, shading indicates the standard error of the log squared difference. Averages over 1085 runs for (a), 872 for (b). See text and main chapter for details.

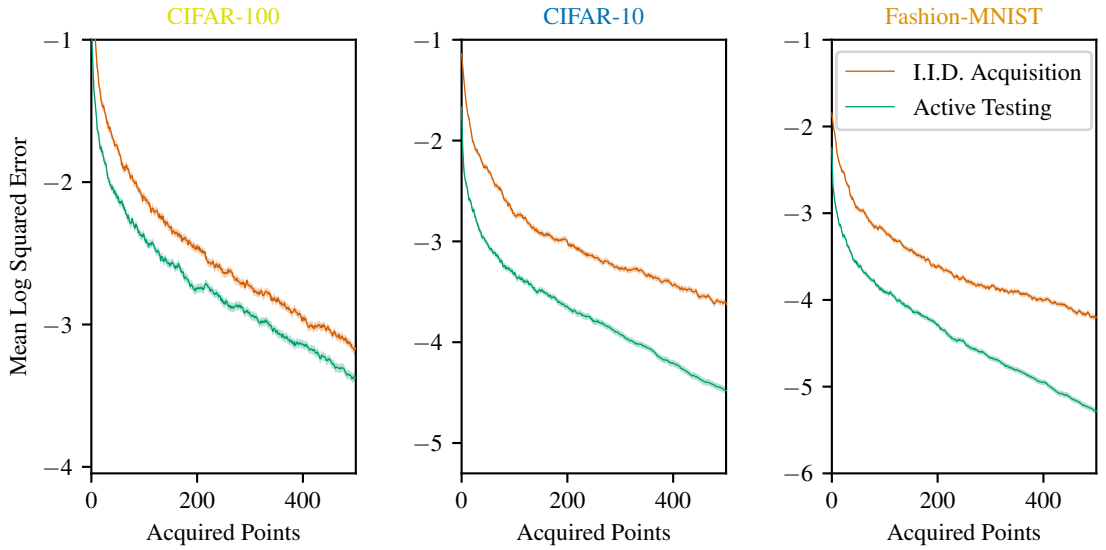


Figure B.4: Fig. 4.6 (a) from the main chapter but now showing the mean of the log squared difference instead of the median. Additionally, shading indicates the standard error of the log squared difference. Averages over 1000 random test set draws. See text and main chapter for details.

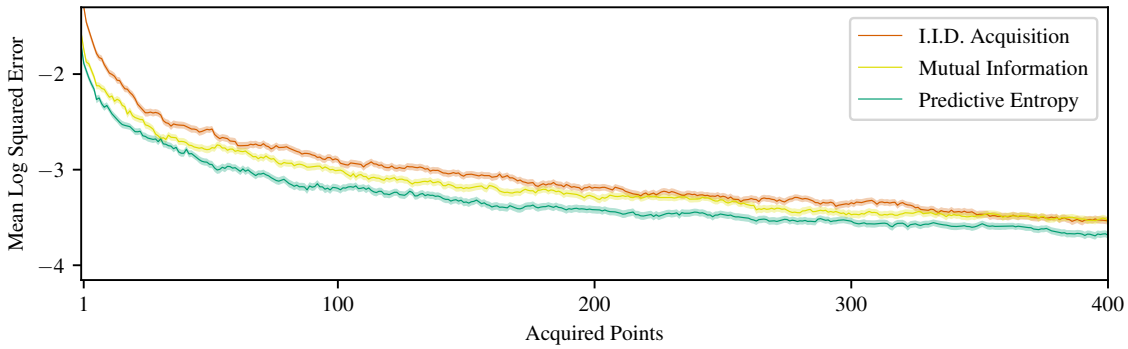


Figure B.5: Fig. 4.8 (b) from the main chapter but now showing the mean of the log squared difference instead of the median. Additionally, shading indicates the standard error of the log squared difference. Averages over 692 runs. See text and main chapter for details.

B.2 Additional Details on Active Testing

B.2.1 Further Details on Clipping

As mentioned briefly in the main chapter, we are clipping the predicted $q(i_m)$ to avoid overly small acquisition values. We do this because $\hat{R}_{\text{LURE}}$ is only unbiased if we have non-zero acquisition probability on *all* points remaining (Farquhar et al., 2021). In practice, we bound the value of the acquisition function from below at α times the acquisition probability assigned by a uniform acquisition strategy, where $\alpha = 0.2$.

B.2.2 Computational Complexity

Two components of active testing contribute significantly to computational complexity: Evaluating the acquisition function on the remaining samples of the test pool and retraining the surrogate on all observed samples.

Acquisition Function Evaluation At each active testing iteration, we require the evaluation of the acquisition function q on all remaining samples in the test pool $\mathcal{D}_{\text{test}}$. At first iteration, N samples remain in the test pool, contributing $\mathcal{O}(N)$ to the computational complexity. At each subsequent iteration m, $\mathcal{O}(N-m)$ additional evaluations are required. For a total of M acquisitions, we can bound the cost of evaluating the acquisition function with $\mathcal{O}(MN)$.

Surrogate Retraining In our experiments in Section 4.5.2, we demonstrate successful active testing when retraining the surrogate every K steps or training the surrogate only once on $\mathcal{D}_{\text{train}}$. However, in active testing practice, label cost may be significantly larger than the cost of retraining. Therefore, we here assume the worst case $K = M$, i.e. retraining the surrogate after each acquisition. This gives maximum sample-efficiency because additional information from newly acquired test labels is directly used to improve surrogate predictions. At first iteration, only $\mathcal{D}_{\text{train}}$ is observed and hence retraining costs will be $\mathcal{O}(|\mathcal{D}_{\text{train}}|)$. As testing progresses, m samples of the test set become available such that retraining costs grow to $\mathcal{O}(|\mathcal{D}_{\text{train}}| + m)$ per iteration. For a total of M acquisitions, retraining

costs therefore scale as $\mathcal{O}(M|\mathcal{D}_{\text{train}}| + M^2)$. Therefore, we can bound the total computational complexity of active testing as $\mathcal{O}(M|\mathcal{D}_{\text{train}}| + M^2 + MN)$, which can be simplified to $\mathcal{O}(M|\mathcal{D}_{\text{train}}| + MN)$ as $M \leq N$ always.

B.3 Experiment Details

B.3.1 Hardware

For the neural network architectures, we use PyTorch (Paszke et al., 2019). We use a mix of NVIDIA RTX 2080, Titan RTX, and K80 GPUs to run the experiments. No experiment requires more than one GPU in parallel. Experiments that do not require GPUs, such as the synthetic data experiments, can be run on CPUs with conventional hardware.

B.3.2 Figures 4.1 to 4.3: Synthetic Data

We now give details for Fig. 4.3 of the main chapter, shown again in Fig. B.1 (columns 1, 4, 5), as well as the additional experiments (columns 3 and 4). Figs. 4.1 and 4.2 use the setup of Fig. B.1 (column 1).

Regression For the Gaussian process data (column 1), we sample $N = 50$ points from a Gaussian process with Matern-Kernel $\nu = 3/2$ and $l = 1$, using the implementation of Pedregosa et al. (2011). For each of the experiments, we sample a different set of points from the Gaussian process prior. The quadratic data (columns 2, 4) are drawn from $f(x) = y^2$. The sinusoidal data (column 3) are drawn from $f(x) = \sin(10x) + x^3$, where the density of the samples in x is non-uniform.

5 points are randomly assigned to the training set, 45 are used for testing. The test-train assignments are randomly redrawn between experiments. For each experiment, we actively evaluate the data with all acquisition functions. We perform a series of 5000 independent experiments. All regression experiments use a Gaussian process surrogate that is retrained after each acquisition on all observed data and has Matern-Kernel $\nu = 3/2$ and $l = 1$.

Two Moons Classification For the two moons dataset (column 5), we randomly sample $N = 500$ points with noise level of 0.1 for each experiment, using the implementation of Pedregosa et al. (2011). 50 points are randomly assigned to the training set, 450 are used for testing. We perform a series of 2500 independent experiments. We use default parameters and implementation of Pedregosa et al. (2011) for the random forest. The surrogate model uses an identical random forest that is retrained after each acquisition on all observed data.

B.3.3 Figure 4.4: Radial BNN/ResNet-18 on MNIST/Fashion-MNIST

Data For each experiment, we sample 250 training and 5000 test points randomly from the combined training and test set of the original datasets. We standardize the dataset using per-channel mean and standard deviation values over all pixels of the training set. We actively acquire 1000 samples from the test set and compute the "full test loss" on all 5000 test points. We perform stratified sampling for both the train/test as well as the train/val splits. We retrain the surrogates after acquiring $(0, 5, 10, 20, 30, 40, 100, 250, 400, 550, 700, 850, 1000)$ test points.

Radial BNN on MNIST We use the code from Farquhar et al. (2021) to implement the Radial BNNs. We use the following hyperparameters, which are default values taken from Farquhar et al. (2021): we use a learning rate of 1×10^{-4} and weight decay of 1×10^{-4} with the ADAM optimizer (Kingma and Ba, 2015), batch size of 64, 8 variational samples during training for the BNN, 100 variational samples during testing, and convolutions with 16 channels. We train for a maximum of 500 epochs with early stopping patience of 5 and use validation sets of size 50. We have not tuned these hyperparameters.

ResNet-18 on Fashion-MNIST For the Resnet-18, we use the default hyperparameter values introduced by DeVries and Taylor (2017): we use a learning rate of 0.1, weight decay of 5×10^{-4}, momentum of 0.9 with an SGD optimizer, and batch size of 128. We use a cosine annealing schedule for the learning rate as provided by

PyTorch. We train for a maximum of 160 epochs with early stopping patience of 20 and use validation sets of size 50. We have not tuned these hyperparameters.

Random Forest Surrogate We use random forests with 100 estimators, split criterion "entropy", and maximum number of features considered at each split proportional to the square root, otherwise using the default parameters and implementation of Pedregosa et al. (2011). Hyperparameters were set with a single grid-search cross-validation on one particular training set.

Training Convergence: Radial BNN on MNIST Training of the BNN takes about 1-2 minutes and the early stop patience usually terminates training after around 50-100 epochs. A single experiment run for the Radial BNN, requiring the training of the original model and retraining of all shown surrogates, takes about 30 minutes. The validation accuracy on the initial 250 points is at about 80-90 percent and grows to about 90-98 percent after observing a total of 1250 points. We perform a total of 1085 runs.

Training Convergence: ResNet-18 on Fashion-MNIST Training of the ResNet takes about 10-40 seconds and the early stop patience sets in around 30-80 epochs. A single run for the ResNet, requiring the training of the original model and retraining of all shown surrogates, takes about 10 minutes. The training validation accuracy on the initial 250 points is at about 68-78 percent and grows to about 80-86 percent after observing a total of 1250 points. We perform a total of 872 runs.

B.3.4 Figure 4.5: ResNet-18 on CIFAR-100

The setup for Fig. 4.5 is a ResNet-18 on CIFAR-100 identical to the experiments from Fig. 4.6 described in the following. The model converges to 72-76 percent accuracy in about 12 minutes and we perform 692 runs.

B.3.5 Figure 4.6: ResNet-18/WideResNet on Fashion-MNIST, CIFAR-10/Cifar-100

Data We now respect the original train and test indices of the dataset. We train a model on the training set and then, for each experiment, perform active testing on a subset of 1000 randomly sampled points from the original test set of size 10 000. We standardize the dataset using per-channel mean and standard deviation values over all pixels of the training set. For CIFAR-10 and CIFAR-100, we apply random crops and horizontal flips to the training data in each epoch. Given the larger training data, we now use validation sets of size 2560. We re-sample test sets for the next experiment, but keep training sets and models constant. We perform stratified sampling for both the train/test as well as the train/val splits. This, we repeat a total of 1000 times.

ResNet-18 Unless otherwise mentioned we use identical hyperparameters as for Fig. 4.4. On all datasets, we train for a maximum of 30 epochs with patience of 5. Again, we do not systematically tune hyperparameters and only make adjustments to accommodate the increased data size compared to Fig. 4.4. The model converges to 93-94 percent accuracy on Fashion-MNIST in about 7 minutes and 92-93 percent accuracy on CIFAR-10 in about 12 minutes.

WideResNet We use a WideResNet of depth 40 and the setup introduced by DeVries and Taylor (2017): we train for 200 epochs, use a learning rate of 0.1, weight decay of 5×10^{-4}, momentum of 0.9 with an SGD optimizer, and batch size of 128. We also use the scheduler of DeVries and Taylor (2017) and decrease the learning rate by a factor of $\gamma = 0.2$ at epochs 60, 120, and 160. The model converges to 78-80 percent accuracy on CIFAR-100 in about 240 minutes.

Ensembles For the deep ensemble of Resnet-18s, we use 10 models for Fashion-MNIST, 5 for CIFAR-10, and 15 for CIFAR-100. For the deep ensemble of WideResnets on CIFAR-100, we use a set of 10 models. The models for the ensembles are identical to the main models, i.e. use the same hyperparameters,

optimizers, and training setup, and only differ in terms of the optima reached from stochastic model initialization and optimization.

Increasing the size of the ensemble did sometimes yield improved results. Following initial experiments, we increased the size of the ensemble for Fashion-MNIST from 5 to 10. However, using ensemble size of 15 on CIFAR-10 did not improve the results noticeably (nor worsen them), so we display the smaller, and computationally cheaper ensemble in the main body.

B.3.6 Figure 4.7

Fig. 4.7 uses the same setup as Fig. 4.6 for CIFAR-10 with main ResNet-18 model and "train ensemble"-style surrogates.

B.3.7 Figure 4.8

For the Radial BNN, Fig. 4.8 uses identical experimental setup as Fig. 4.4, except that the model is trained for a maximum of 30 epochs (patience 5) and we use validation sets of size 1280, to account for the increase in data. For the Fashion-MNIST data, we concatenate the original train and test datasets, from which we then sample 50 000 points without replacement for the training dataset and 10 000 for the test dataset. We acquire a total of 1000 points from the test dataset but compute the "full test set loss" on all 10 000 points. Splits are stratified by class labels. We redraw train and test splits, and retrain models between runs. For each experiment, the model reaches about 88-92 percent validation accuracy in about 8 minutes. We perform a total of 962 runs.

B.3.8 Figure B.2: Radial BNN on MNIST.

For the Radial BNN, Fig. B.2 uses identical experimental setup as Fig. 4.4 (a), except that the model is trained for a maximum of 50 epochs (patience 5) and we use validation sets of size 1280 to account for the increase in data. For the MNIST data, the setup is identical to Fig. 4.8. The model reaches about 98 percent validation accuracy in about 7-10 minutes. We perform a total of 992 runs.

B.4 Comparison to Active Risk Estimation by Sawade et al.

We here provide a brief introduction to "Active Risk Estimation" (ARE) by Sawade et al. (2010) as well as an empirical comparison of their approach to ours. Similar to us, Sawade et al. (2010) actively acquire labels from a test pool of unlabeled samples, aiming to obtain a sample-efficient estimate of the empirical test risk. However, unlike us, they do not fully accommodate the pool-based setting. Additionally, Sawade et al. (2010) rely only on the original model to approximate outcomes $Y \mid \boldsymbol{x}$, while we have shown that adaptive surrogate models can be crucial for achieving label-efficiency. Moreover, as we show below, active testing outperforms ARE in practice.

B.4.1 Background

Sawade et al. (2010) derive an "optimal acquisition function"

$$q^*(\boldsymbol{x}) \propto p(\boldsymbol{x}) \sqrt{\int \left[\mathcal{L}(f_\theta(\boldsymbol{x}), y) - r\right]^2 p(y \mid \boldsymbol{x}) \, \mathrm{d}y}, \tag{B.1}$$

where $r = \mathbb{E}\left[\mathcal{L}(f(\boldsymbol{X}), Y)\right]$ is the true model risk. For the pool-based setting, they obtain an equivalent "optimal acquisition function" by setting $p(\boldsymbol{x}_{i_m}) = \frac{1}{M}$ and approximating r with the empirical test risk $\hat{R}_{\text{IID}}$ over the entire test pool. As $\hat{R}_{\text{IID}}$ is still unknown, Sawade et al. (2010) approximate $\hat{R}_{\text{IID}}$ using the original model's mean predictions $f_\theta(\boldsymbol{x})$ to approximate the true outcomes, giving

$$\hat{R}_{\text{IID},\theta} = \frac{1}{M} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{test}}} \int \mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y}) p(\boldsymbol{y} | \boldsymbol{x}; \theta) \, \mathrm{d}\boldsymbol{y} \tag{B.2}$$

$$q^*(\boldsymbol{x}) \propto \sqrt{\int \left[\mathcal{L}(f_\theta(\boldsymbol{x}), y) - \hat{R}_{\text{IID},\theta}\right]^2 p(y \mid \boldsymbol{x}; \theta) \, \mathrm{d}y}. \tag{B.3}$$

They do not consider the concept of a surrogate model.

Plugging mean squared error loss and Gaussian likelihoods into Eq. (B.3), they derive the acquisition function

$$q(\boldsymbol{x}) \propto \sqrt{3\sigma_{\boldsymbol{x}}^4 - 2\hat{R}_{\text{IID},\theta}\sigma_{\boldsymbol{x}}^2 + \hat{R}_{\text{IID},\theta}^2}, \tag{B.4}$$

where $\sigma_{\boldsymbol{x}}^2$ is the total predictive variance (aleatoric and epistemic uncertainty) of the original model's prediction at $\boldsymbol{x}$.

For risk estimation, ARE relies on a standard importance sampling estimator

$$\hat{R}_{\text{IS}} = \frac{1}{\sum_{m=1}^{M} q(\boldsymbol{x}_{i_m})} \sum_{m=1}^{M} \frac{1}{q(\boldsymbol{x}_{i_m})} \mathcal{L}(f_\theta(\boldsymbol{x}_{i_m}, y_{i_m})), \tag{B.5}$$

where the uniform $p(\boldsymbol{x}_{i_m})$ cancels from the weights. Unlike $\hat{R}_{\text{LURE}}$, $\hat{R}_{\text{IS}}$ requires sampling *with replacement* to obtain unbiased estimates. In other words, ARE does not remove samples from the test pool after querying them, and often, test points will get sampled repeatedly.

B.4.2 Empirical Comparison.

We compare both theoretically optimal behavior as well as observed practical performance of our active testing and ARE. For this, we take the familiar setup from Fig. B.1 (column 1).

Optimal Behavior We have seen that active testing can, in a theoretically optimal setup, lead to single-sample, zero variance estimates of the test loss (cf. Fig. 4.8 (a)). What is the ideal behavior that ARE can obtain? To test this, we apply Eq. (B.1) with the oracle empirical test risk $\hat{R}_{\text{IID}}$ and the true $y \mid \boldsymbol{x}$ (which is a δ-distribution since we generate data without noise). This constitutes the best case scenario for ARE performance.

As Fig. B.6 (a) shows, "optimal ARE" can improve over naive IID acquisition but does not show the same desirable single-sample optimal behavior as active testing. Note that ARE extends beyond the test set size of 45 because it does not naturally converge due to sampling with replacement. In Fig. B.6 (b) we confirm that naive extensions of ARE to sampling *without replacement* are not successful: (i) Simply using ARE but sampling without replacement does not work; (ii) When additionally using $\hat{R}_{\text{IID}}$ instead of $\hat{R}_{\text{IS}}$, we converge to the empirical test risk but observe a bias and increased variance over naive IID acquisition at earlier acquisition steps.

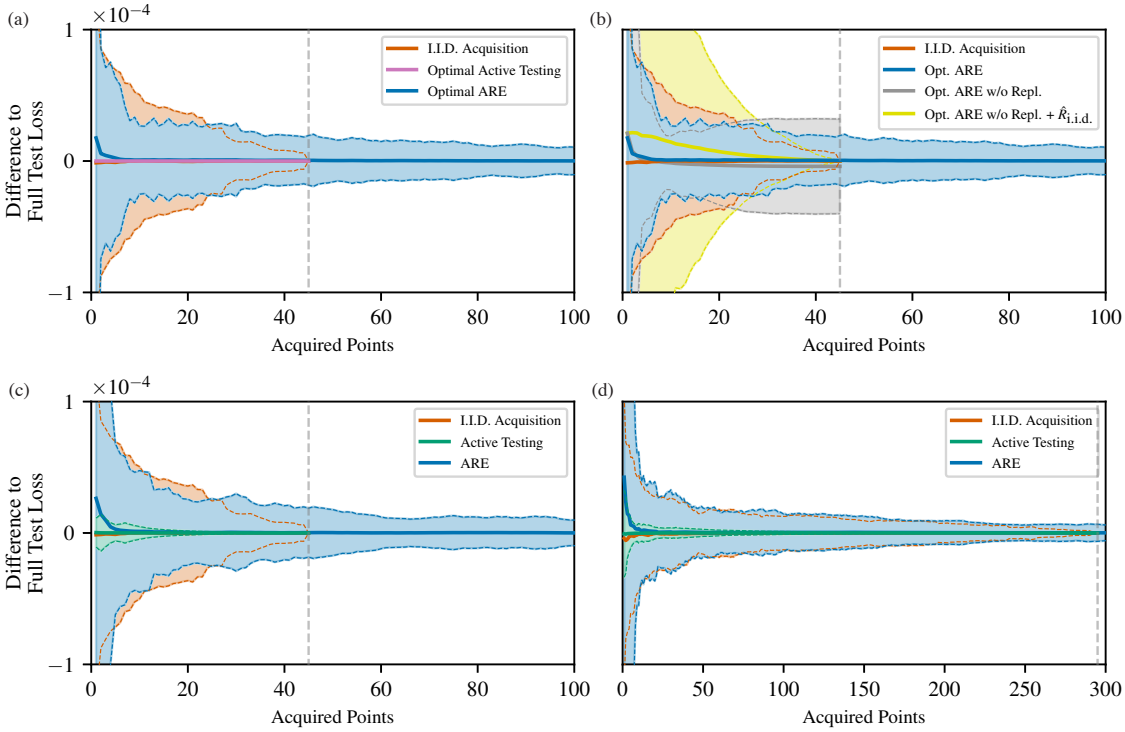


Figure B.6: Comparison of Active Testing to Active Risk Estimation (ARE) by Sawade et al. (2010). (a) While active testing can yield single-sample zero-variance estimates in an idealized scenario, ARE cannot because it does not fully accommodate the pool-based setting. (b) Naive extensions of ARE to sampling without replacement are unsuccessful. (c) In practice, ARE yields lower performance than active testing. (d) As the test set grows, the gap between ARE and active testing will get smaller as sampling with replacement matters less. Dashed grey line demarcates the number of samples in the test pool. Solid lines show means, dashed lines standard deviations over 7000 (a–c) / 2000 (d) runs.

Performance in Practice We now investigate behavior of ARE in practice—where the oracle risk is not available and we rely on Eq. (B.3) instead. Fig. B.6 (c) shows that while ARE can increase sample-efficiency over naive IID acquisition in practice, it is clearly outperformed by active testing. For Fig. B.6 (d), we increase the size of the test set from 45 to 295. We suspect that further increases in test set size will shrink the gap between ARE and active testing without surrogates, because sampling with replacement should become less important. However, still, active testing clearly outperforms ARE: active testing also varies substantially in the acquisition strategy it uses (in particular allowing the use of surrogates), with these result suggesting its approach is clearly preferable.

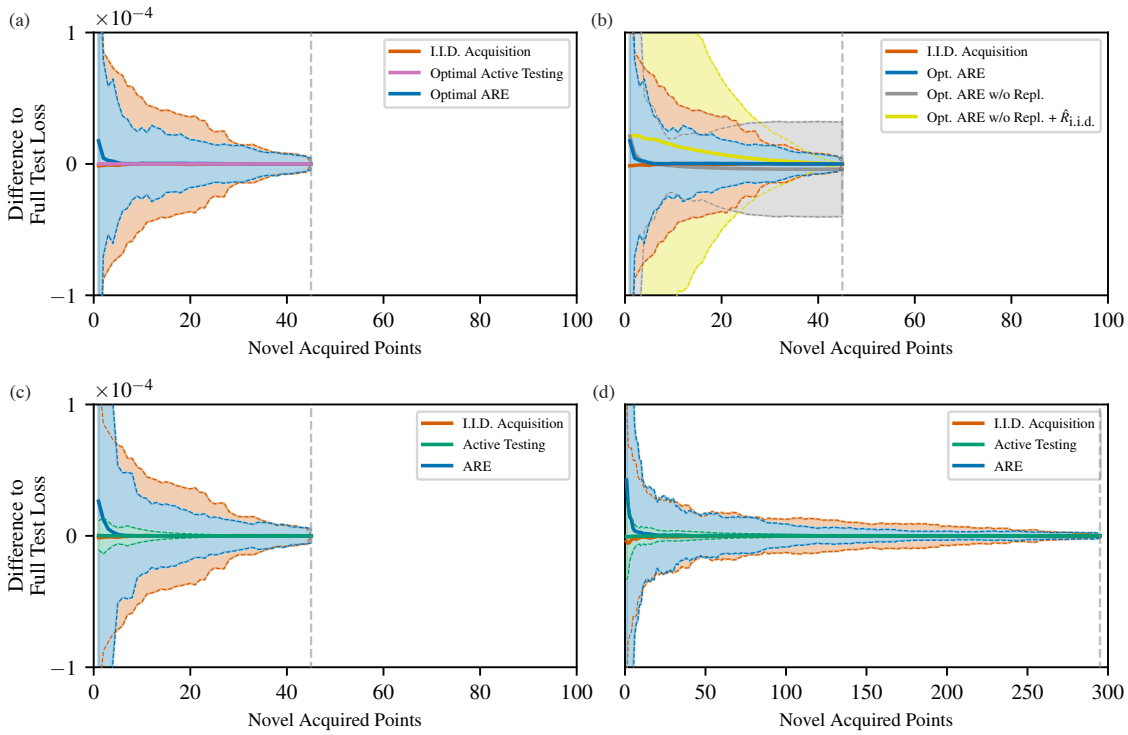


Figure B.7: Comparison of Active Testing and Active Risk Estimation. Identical to Fig. B.6, except that now, acquisition steps for ARE are only counted if ARE queries lead to novel label acquisitions. Displaying results this way slightly improves ARE's performance but does not change our conclusions.

Redefining Acquisitions Steps Sampling with replacement is a sub-optimal strategy for methods applied in pool-based settings. As mentioned above, sampling with replacement in ARE leads to the same samples being acquired multiple times.[2] For maximum fairness, we now only increase the "acquisition step" counter by one if the sampling with replacement process leads to an acquisition that has not been previously made, i.e. if the acquisition is *novel*. This is somewhat justified in practice because we assume that acquiring novel labels is much more expensive than acquisition function evaluations. We display the results with "rescaled x-axis" for ARE in Fig. B.7. While the performance of ARE appears slightly improved, our overall conclusions are unaffected.

[2]For Fig. B.6 (c), ARE takes (9.8 ± 3.1) times as many queries as there are samples in the test pool. Sampling with replacement is not a desirable strategy for pool-based active model evaluation.

C

Active Surrogate Estimators

Contents

C.1 Additional Experiments

C.1.1 Comparison of Acquisition Behavior

We here investigate *how* XWED outperforms all competing acquisition strategies in the experiments of Section 5.6.1. As a reminder, in the distribution shift setup the model f is not exposed to any samples of class "7" during training, while the test set contains 7s in their normal proportion. Hence, predictions at samples with class 7 should contribute significantly to the risk of f on the test distribution.

However, the overall number of training points for f is relatively small such that contributions from other classes to the risk are not negligible.

In Fig. 5.2, we examine the probability of acquiring a "7" as a function of the number of acquired points for XWED, BALD, expected loss acquisition, and a uniformly at random sampling strategy (MC). We see that XWED initially focuses on 7s but then diversifies. The baselines always prioritize 7s or never do. The behavior of XWED is preferable: we are initially unsure about the loss of these points, but once the loss is well characterized for the 7s we should explore other areas as well. Note also that this highlights the benefit of ASEs over LURE: LURE, which requires expected loss acquisition, inefficiently needs to keep giving preferences to 7s even once the loss for these is well characterized.

C.1.2 Constant π Fails for Distribution Shift.

We investigate if retraining of the auxiliary model π is necessary to achieve competitive active testing performance in the distribution shift scenario. Here, we train π once on the original training data and then keep it constant throughout testing, not updating π with any information from the test data. ASE and LURE thus use the exact same π. When π is constant, the ASE *estimate* and the LURE *proposal* are constant. For LURE, we acquire by sampling from Eq. (4.6) as usual. Fig. C.1 (a) shows that, for both ASE- and LURE-based active testing, naive MC cannot be outperformed when π is fixed. For this distribution shift scenario, it therefore seems that a fixed π is neither informative enough as a proposal for LURE, nor does it allow for good ASE estimates. This justifies our focus on *adaptive* approaches for active testing.

C.1.3 Additional Acquisition Strategies for ASEs

For completeness, we here also investigate the performance for ASE when performing *stochastic* acquisition from the XWED and BALD scores. These are the exact acquisition strategies for which LURE suffered high variance in Fig. 5.3. In Fig. C.1 (b), we observe that ASE continues to perform well for both these

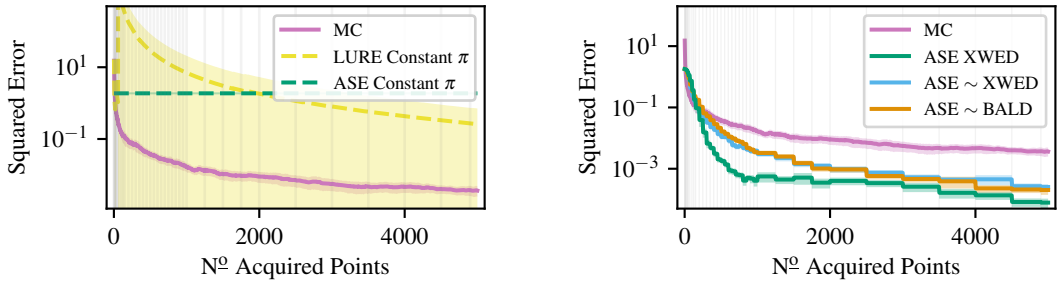


Figure C.1: (a) If π is constant, both ASE and LURE do not outperform naive MC in the distribution shift scenario. This result highlights the importance of the *adaptive* nature of both ASE- and LURE-based active testing. (b) Additional acquisition functions for ASE in the distribution shift scenario: while deterministic acquisition from XWED performs best, ASEs also give good performance when acquiring *stochastically* from XWED or BALD scores. This is unlike LURE, which is highly sensitive to the correct choice of acquisition function. We display means over 100 runs and shading indicates two standard errors (too small to be visible for some of the lines). The vertical grey lines mark iterations at which we retrain the secondary model.

acquisition strategies, although they do not outperform our default strategy that uses deterministic acquisition of the maximal XWED scores.

C.1.4 Estimating Accuracy with ASEs

ASEs are compatible with arbitrary loss functions $\mathcal{L}$. For 0-1-Loss, $\mathcal{L}(f(\boldsymbol{x}), y) = \mathbf{1}[f(\boldsymbol{x}) = y]$, ASEs can be used to estimate the accuracy of the main model. We perform these experiments using the same setup as in Section 5.6.3. Figure C.2 demonstrates that ASEs continue to outperform all other baselines for the task of accuracy estimation. The accuracies of the main model in these experiments are $(73.31 \pm 0.93)\%$ for CIFAR-100, $(91.07 \pm 0.62)\%$ for CIFAR-10, and $(93.10 \pm 0.53)\%$ for Fashion-MNIST.

C.1.5 Reduced Training Sets for Experiments of Section 5.6.3

We here investigate a variation of the experiments in Section 5.6.3 that reduces the size of the training set to 10 000 samples. This reduces the quality of the surrogate model and thus makes the problem more challenging for ASEs. Despite this, Fig. C.3 demonstrates that ASEs continue to outperform all baselines. Again, note that the ASE does not use any test labels here and is therefore constant.

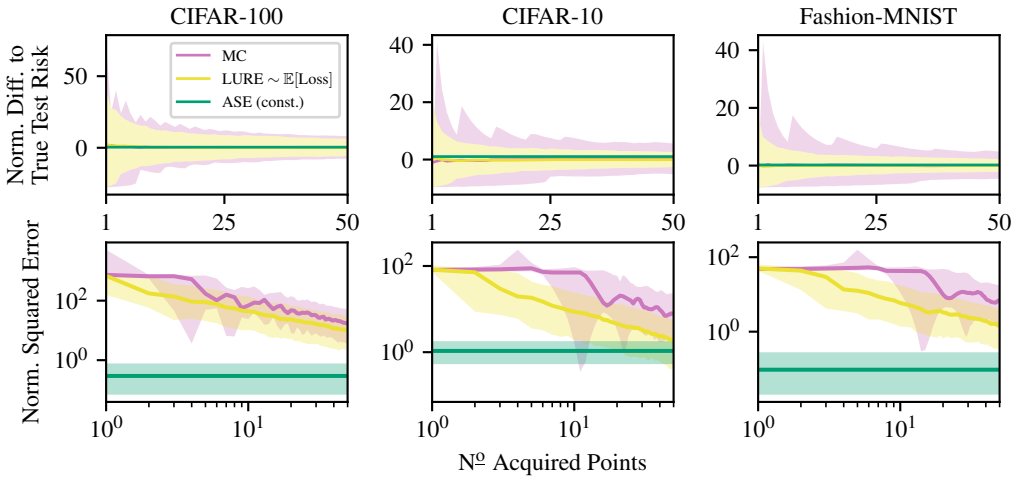


Figure C.2: Variant of the experiments of Section 5.6.3 where we estimate the *accuracy* of the main model. We display medians over 1000 random test sets and 10/90 % (top) and 25/75 % (bottom) quantiles (top: too small to be visible for ASE).

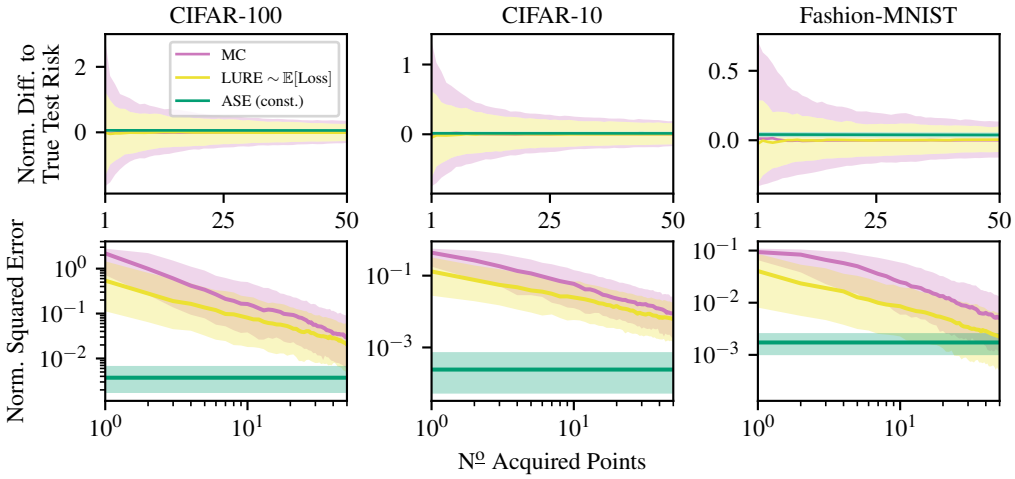


Figure C.3: Variant of the experiments of Section 5.6.3 where the training set size is reduced to 10 000 samples. We display medians over 1000 random test sets and 10/90 % (top) and 25/75 % (bottom) quantiles (top: too small to be visible for ASE).

Retraining of the ASE surrogate on the newly observed test data (with labels) will be necessary for ASEs to be able to continue to improve over the baselines at larger number of acquired test labels.

C.2 Additional Figures

In the main chapter, we display both median and mean squared errors across the different figures to provide a variety of perspectives on ASEs. Here, we provide complimentary versions: for figures that show median behavior in the main chapter,

we here show means, and vice versa.

See Fig. C.4, Fig. C.5, Fig. C.6, and Fig. C.7 for complimentary versions of the plots from the experimental evaluations in Section 5.6 and Appendix C.1.

For ASE, there are no significant differences between the mean and median behavior. In contrast, LURE occasionally does suffer from high variance (mean behavior), e.g. for CIFAR-10 in Fig. C.6. As discussed in the main chapter, this happens when the weight assigned by the proposal diverges strongly from the observed outcome for individual observations. This does not happen for ASE because, unlike LURE, we do not weight observations against predictions from the surrogate.

We also include versions of the main figures of the chapter that normalize the squared errors of the estimators by the squared true test loss of the main model f on the full test set. See Fig. C.8, Fig. C.9, and Fig. C.10 for these plots. For the experiments in Section 5.6.1 and Section 5.6.2, the cross-entropy loss of the main model is 1.51 ± 0.18 (average over 100 runs) and the accuracy is $(85.63 \pm 0.25)\%$. For the experiments of Section 5.6.3, the cross-entropy losses are 0.961 ± 0.039 for CIFAR-100, 0.289 ± 0.021 for CIFAR-10, and 0.172 ± 0.015 for Fashion-MNIST. The accuracies are 72.54% for CIFAR-100, 90.92% for CIFAR-10, and 92.05% for Fashion-MNIST. (We provide standard deviations on accuracies for the experiments of Appendix C.1.4.)

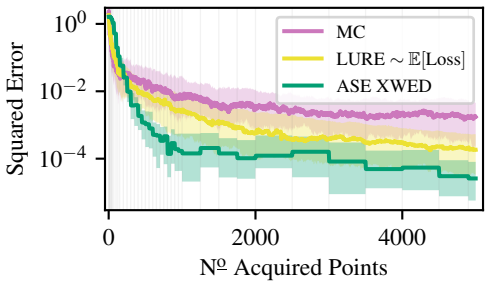


Figure C.4: Version of Fig. 5.1 with medians and quantiles instead of means and standard errors. Convergence of the squared error for the distribution shift experiment. Shown are median squared errors and the shading indicates the 25/75 % quantiles over 100 runs. We see that ASE provides significant improvements over both LURE and MC. The vertical grey lines mark where we retrain the secondary model.

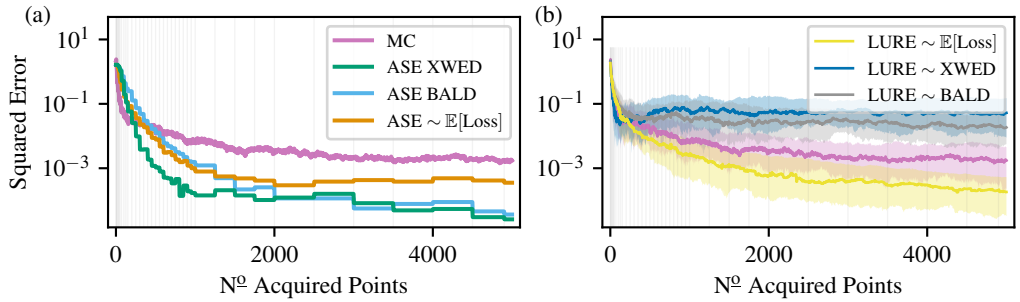


Figure C.5: Version of Fig. 5.3 with median squared errors. (a) While XWED performs best, ASE outperforms MC for all investigated choices of acquisition functions. Shown are median squared errors over 100 runs, and we omit quantile shading to aid legibility. (b) LURE suffers high error when not acquiring from expected loss and instead using a different acquisition strategy. Shown are median squared errors and the shading indicates 25/75 % quantiles over 100 runs. We retrain π at steps marked with grey lines. The "$\sim$"-symbol indicates stochastic acquisition.

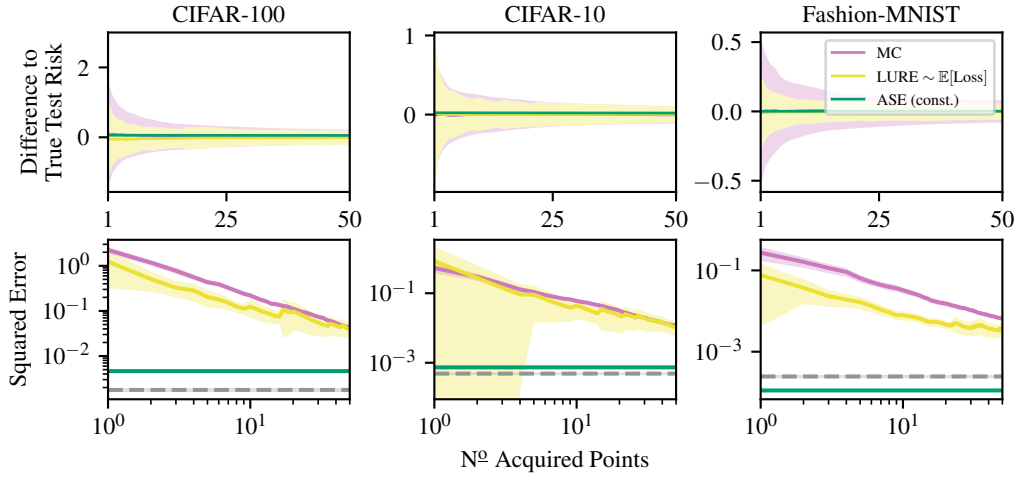


Figure C.6: Version of Fig. 5.4 with means and standard errors. We display means over 1000 random test sets and shading indicates standard deviations for the differences (top, too small to be visible for ASE) and two standard errors for the squared error (bottom, too small to be visible for ASE and naive MC). Active Testing of ResNets on CIFAR-100, CIFAR-10, and FashionMNIST without retraining the auxiliary model π. Despite not observing any test labels, ASE again improves upon LURE. The grey line indicates the lower bound on the error from the finite size of the pool.

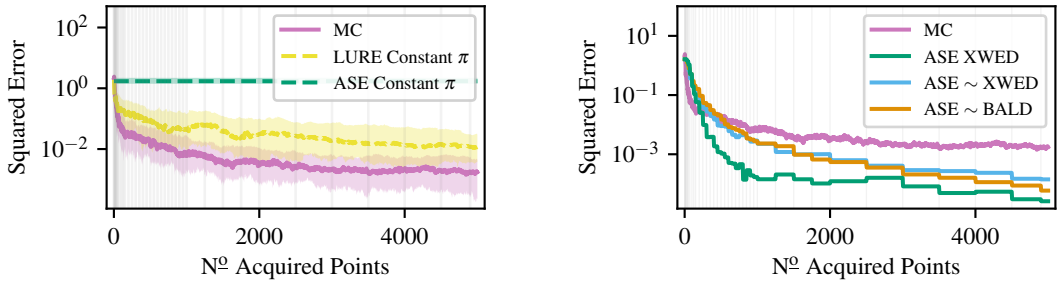


Figure C.7: (a) Version of Fig. C.1 (a) with medians and quantiles instead of means and standard errors. Convergence of the squared error for the distribution shift experiment with constant secondary model π. Shown are median squared errors and the shading indicates the 25/75 % quantiles over 100 runs. Again, we see naive MC is not outperformed if π is constant. (b) Version of Fig. C.1 (b) with medians instead of means. Comparison of *stochastic* acquisition strategies for ASE on the distribution shift experiment. Shown are median squared errors over 100 runs and we omit quantiles for reasons of legibility. The vertical grey lines mark where we retrain the secondary model.

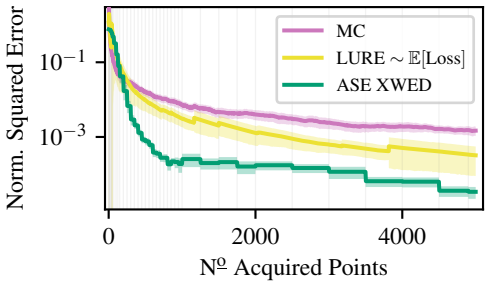


Figure C.8: Version of Fig. 5.1 with squared error of the estimates normalized by the true value of the test error.

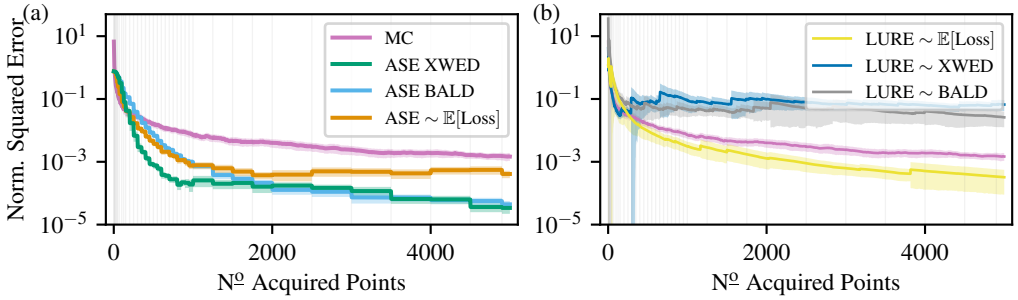


Figure C.9: Version of Fig. 5.3 with squared error of the estimates normalized by the true value of the test error.

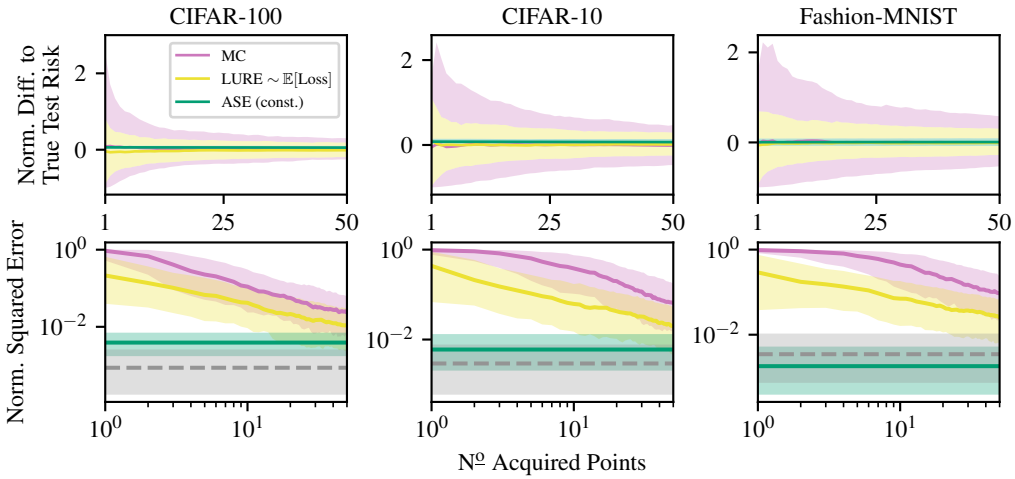


Figure C.10: Version of Fig. 5.4 with differences and squared errors of the estimates normalized by the true value of the test error.

C.3 Experiment Details

C.3.1 Experiment Definition

In this section, we provide a more detailed explanation of how we combine different estimators and acquisition functions in the experimental evaluation in Section 5.6. For our purposes, there are two ingredients to a method for model evaluation: the estimator and the acquisition strategy.

In terms of estimators, we compare the naive MC estimator, $\hat{R}_{\text{IID}}$, Eq. (4.2), and the LURE estimator, $\hat{R}_{\text{LURE}}$, Eq. (4.3), against the ASE estimator, $\hat{R}_{\text{ASE}}$, Eq. (5.2).

For the MC estimator, the acquisition of labels from the remaining samples in the test pool is always uniformly at random, so in the figures in Section 5.6, we simply write "MC" to denote this naive baseline. For any other acquisition scheme, $\hat{R}_{\text{IID}}$ would no longer be unbiased. However, for ASE and LURE, we have the option to use a variety of acquisition functions a. In this paper, we consider XWED, Eq. (5.4), BALD, Eq. (5.3), and $\mathbb{E}$[Loss], Eq. (4.6) as acquisition functions. For all of the these acquisition functions, at any iteration m, we evaluate the acquisition function on all remaining samples $\mathcal{U}$ in the test pool, obtaining a set of acquisition scores $\{a(x_i)\}_{i \in \mathcal{U}}$.

Given these scores, there are two ways in which we can determine the next acquisition: deterministic and stochastic acquisition. In deterministic acquisition,

we acquire a label for the sample x_{i_m} remaining in the test pool with the largest acquisition score $i_m = \arg\max_i a(x_i)$. For stochastic acquisition, we choose the next point to acquire by sampling from the distribution over acquisition scores: we define a probability mass function over the indices as $p(i) = a(x_i) / \sum_{i \in \mathcal{U}} a_i$ and then sample an index $i_m \sim p(i)$ for acquisition. For the LURE estimator, acquisition needs to be stochastic for the importance sampling estimator to remain unbiased, while ASEs are compatible with both stochastic and deterministic acquisition.

In the experiments in Section 5.6, we add a "$\sim$" prefix for experiments with stochastic acquisition, and we add no prefix for deterministic acquisition. For example, "LURE $\sim$ BALD" is an experiment that uses the LURE estimator and stochastically samples acquisitions from the BALD scores, while "ASE XWED" denotes a run that uses the ASE estimator and deterministically acquires labels for samples with the highest XWED score in each round.

C.3.2 Computing XWED and Related Quantities

To give a practical example of how to compute some of the quantities related to the ASE and active testing, we now present a worked through example for the case in which the main model f is a neural network and the surrogate π a deep ensemble, both trained for a classification task.

Here, both $f(\boldsymbol{x})$ and $\pi(y \mid \boldsymbol{x})$ are classifiers whose outputs are vectors of dimensionality C, where C denotes the number of classes. Each element i of these vectors, represents the predicted probability that the observed class y is equal to i. The quantity we wish to estimate is the average negative log-likelihood on the test set, also known as the cross-entropy loss of the test set. Our loss function is thus $\mathcal{L}(f(\boldsymbol{x}), y) = -\log f(\boldsymbol{x})_y$, i.e. the negative log probability the model f assigns to the true class.

For classification tasks, we can evaluate expectations with respect to outcomes Y exactly by enumerating all possibilities. Further, for the deep ensemble surrogate π, we model the posterior distribution of the deep ensemble surrogate parameters as $\pi(\boldsymbol{\theta}) = \frac{1}{E} \sum_{e=1}^{E} \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_e)$, where δ is the Dirac delta and $e \in \{1, \ldots, E\}$ enumerates

the ensemble components. Together, this simplifies the XWED acquisition function to simple sums over class labels and ensemble components,

$$\begin{aligned} \mathrm{XWED}(\boldsymbol{x}) = &\sum_{y=1}^{C} -\pi(y \mid \boldsymbol{x}) \mathcal{L}(f(\boldsymbol{x}), y) \log \pi(y \mid \boldsymbol{x}) + && \text{(C.1)} \\ &\frac{1}{E} \sum_{e=1}^{E} \pi(y \mid \boldsymbol{x}, \boldsymbol{\theta}_e) \mathcal{L}(f(\boldsymbol{x}), y) \log \pi(y \mid \boldsymbol{x}, \boldsymbol{\theta}_e), && \text{(C.2)} \end{aligned}$$

where, lastly, we note that a posterior predictive distribution for deep ensembles can be obtained as the average prediction of the ensemble components,

$$\begin{aligned} \pi(y \mid \boldsymbol{x}) &= \int \pi(y \mid \boldsymbol{x}, \boldsymbol{\theta}) \pi(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} && \text{(C.3)} \\ &= \frac{1}{E} \sum_{e=1}^{E} \int \pi(y \mid \boldsymbol{x}, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_e) \, \mathrm{d}\boldsymbol{\theta} = \frac{1}{E} \sum_{e=1}^{E} \pi(y \mid \boldsymbol{x}, \boldsymbol{\theta}_e). && \text{(C.4)} \end{aligned}$$

C.3.3 Training Setup

Evaluation with Distribution Shift We follow Chapter 4 in the setup of the radial BNN, using code and default hyperparameters provided by Farquhar et al. (2021): we employ a learning rate of 1×10^{-4} and set the weight decay to 1×10^{-4}, in combination with the ADAM optimizer (Kingma and Ba, 2015) and a batch size of 64. We use 8 variational samples during training and 100 variational samples during testing of the BNN. We use convolutional layers with 16 channels. We train for a maximum of 500 epochs with early stopping patience of 10, and we use validation sets of size 500. We further use the validation set to calibrate predictions via temperature scaling.

Evaluation of CNNs for Classification We follow Chapter 4, which follows DeVries and Taylor (2017), for the setup of the ResNet-18 for the CIFAR-10 and Fashion-MNIST datasets. Concretely, we set the learning rate to 0.1, the weight decay 5×10^{-4}, use momentum of 0.9 with an SGD optimizer and batch size of 128. We anneal the learning rate using a cosine schedule. We use early stopping on a validation set of size 5000, with patience of 20 epochs, and we set the maximum number of epochs to 160. We use the validation set to calibrate predictions via temperature scaling.

Similarly, we follow Chapter 4 for the setup of the WideResNet on CIFAR-100: we set the depth to 40 and again follow DeVries and Taylor (2017) to set hyperparameters. Specifically, we train for 200 epochs, setting the learning rate to 0.1, the weight decay to 5×10^{-4}, and the momentum to 0.9. We use the SGD optimizer with a batch size of 128. We decrease the initial learning rate by a factor of 0.2 after 60, 120, and 160 epochs. Again, we use the validation set to calibrate predictions via temperature scaling.

C.4 Computational Complexity

The computational complexity of ASEs can be split into two components: the computational complexity of the ASE estimate and the computational complexity of the acquisition process.

The ASE estimate requires us to predict with the surrogate model on each of the N points in the test pool, cf. Eq. (5.2), leading to a computational complexity of $\mathcal{O}(N)$.

During acquisition, at each iteration m, we need evaluate the acquisition function on all $N - m$ remaining samples in the pool. This is repeated for all M iterations leading to an overall computational complexity of $\mathcal{O}(MN)$. Additionally, we have to consider the computational complexity of re-training the surrogate model between iterations. We assume, for worst case complexity, that the surrogate is retrained in each of the M iterations. Further, we assume that the complexity of training the surrogate is proportional to the size of the dataset. If we assume an initial training set of size $|\mathcal{D}_{\text{train}}|$ for the surrogate, then in each testing iteration m, we incur additional computational complexity of $\mathcal{O}(m + |\mathcal{D}_{\text{train}}|)$, or $\mathcal{O}(M \cdot (M + |\mathcal{D}_{\text{train}}|))$ over all iterations. In total, this gives the computational complexity of ASEs as $\mathcal{O}\left(N + M \cdot (N + M + |\mathcal{D}_{\text{train}}|)\right)$. Given that $M \leq N$ always, this can be simplified to $\mathcal{O}\left(M \cdot (N + |\mathcal{D}_{\text{train}}|)\right)$.

Similarly, for Chapter 4, we derive the computational complexity of active testing with LURE as $\mathcal{O}(M \cdot (N + |\mathcal{D}_{\text{train}}|)))$. Therefore both ASE- and LURE-based active testing have identical big-$\mathcal{O}$ complexity. Trivially, the naive Monte Carlo baseline

simply has a much lower complexity of $\mathcal{O}(M)$ than either active testing approaches. We note that, while naive Monte Carlo is cheaper for a given number of acquisitions M, the expense of ASEs and LURE is justified in the active testing scenario by the assumption that these labels are expensive to acquire. Importantly, the *labeling* cost of ASE and LURE is exactly the same as MC: $\mathcal{O}(M)$.

D

In-Context Learning of Label Relationships in LLMs

Contents

D.1 Can Prompts Help ICL Learn Flipped Label Relationships?

In this section, we explore if *prompts* can be used to overcome the plateauing ICL performance when default labels are flipped. Before the in-context examples, we insert prompt strings that inform the LLM of the flipped label relationship in some fashion, and should thus help the LLM adjust to it during ICL. These prompts could fundamentally change few-shot ICL behavior. In fact, one can think of the in-context learner as the union of LLM and prompt, where so far the prompt was

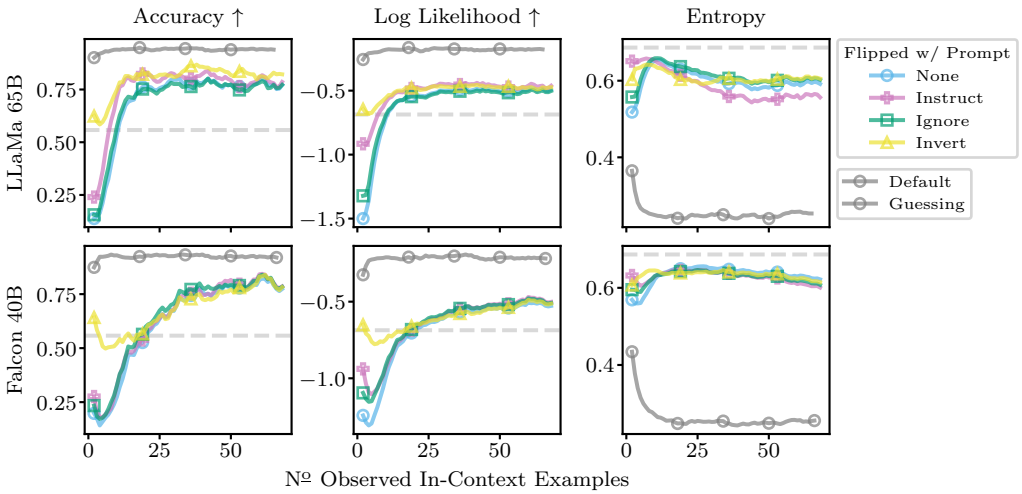


Figure D.1: Prompted few-shot ICL with **flipped labels** on **SST-2**. Some prompts are able to improve ICL on flipped labels compared to not using a prompt as before (label none in the figure). However, improvements have no lasting effect: performance at larger context sizes does not improve and still plateaus short of the default scenario (solid grey line). We average over 100 random subsets and then additionally apply moving averages (window size 5) for clarity.

simply left empty. There could exist prompts that help ICL learn the flipped label relationship better than without them, or as well as in the default scenario. Note that, regardless of the outcome here, NH2 remains rejected as ICL should not need to rely on a prompt to correctly consider in-context observations. Nevertheless, for the 'prompted few-shot ICL' setup, NH2 should be reconsidered.

We explore the following three prompts: 'In the following ...', (Instruct Prompt) '...negative means positive and positive means negative' (here for SST-2 and adapted to other tasks), (Ignore Prompt) '...ignore all prior knowledge', and (Invert Prompt) '...flip the meaning for all answers'.

Observations & Discussions Fig. D.1 gives results for the prompted few-shot ICL setup for LLaMa-65B and Falcon-40B on SST-2 and the original publication in Kossen et al. (2024) additionally contains results for our largest models across all tasks. However, prompting is most successful for the scenarios in Fig. D.1, making this the most interesting result to study NH2. In particular, prompting has a surprisingly weak effect for LLaMa-2-70B. In Fig. D.1 we observe that prompts, in particular the instruct and invert prompts, can help improve ICL performance.

However, it seems that the positive impact from prompting is restricted to an initial boost at small in-context datasets sizes. We then sometimes observe a 'dip' in performance, which could indicate ICL forgetting about the prompt. At large context sizes, none of our prompts have any advantage, and flipped label performance again plateaus short of performance for the default label setup. ***Therefore, we reject NH2 for the prompted ICL variations that we study.***

It is possible that there exist prompts—that we have not found—for which we cannot reject NH2. However, we are sceptical these prompts exist for the models we study, as their behavior at large context sizes is strikingly similar across all prompts we investigate. For more capable LLMs, we suspect it may be possible to obtain a zero-shot performance on the flipped scenario that is equal to the zero-shot of the default scenario, i.e. the prompt leads to the LLM perfectly flipping all its zero-shot predictions. However, we are unsure if, in addition to flipping zero-shot predictions, such prompts would also improve *ICL* on flipped label observations to be as good as in the default scenario.

D.2 Evaluation Approach for Cheap In-Context Learning Dynamics

In this section, we suggest a – to the best of our knowledge – novel way of evaluating ICL that gives performance metrics at all in-context dataset sizes in a single forward pass without incurring additional cost. We start by introducing the notation necessary to formalize few-shot ICL in LLMs.

Dataset to Input String The few-shot task is defined by a dataset $\mathcal{D} = \{(S_i, Y_i)\}_{i=1}^{N}$, where $S_i \in \mathcal{T}^{d_{S_i}}$ are input sentences from which to predict the associated labels $Y_i \in \mathcal{T}^{d_{y_i}}$, and $\mathcal{T}^v$ are text strings of length v. A *verbalizer* $V(S, Y)$ takes a sentence–label pair and maps it to an *example*, e.g. the sequence "I am happy" and label "positive" are verbalized as "Sentence: I am happy\n Label: positive\n". We also define a *query verbalizer* $V_q(S)$ that maps a test query S to a query example, e.g. "I am sad" is mapped to "Sentence: I am sad\n Label:", such

that the next-token prediction of an LLM will be encouraged to predict the label for the query. We apply the verbalizer to the entire dataset set and concatenate its output to obtain the *context* $\mathcal{C} = \oplus_{i=1}^{N} V(S_i, Y_i)$. Finally, we concatenate context $\mathcal{C}$ and verbalized query $V_q(S)$, where S is a sentence drawn from a separate test set, to obtain the input to the language model, $I = \mathcal{C} \oplus V_q(S) \in \mathcal{T}^{d_I}$.

Input String to Tokens The input I is *tokenized* before it can be processed by the language model. The tokenizer, $T(I) = (X_1, \ldots, X_M)$, maps an input sequence I to a sequence of integers, or tokens, $X_i \in (1, \ldots, D)$, where D is the vocabulary size, i.e. the number of unique tokens. We keep track of which token positions correspond to labels, $\mathcal{L} = (l_1, \ldots, l_N)$, e.g. the indices of the tokens immediately following the string "Label:" in the above example.

Tokens to Predictions In the following, we use capital letters to denote random variables and lower-case letters for their realizations. Here, we describe the behavior of *decoder-only* language models (Liu et al., 2018; Radford et al., 2018), a popular architecture choice for LLMs. Given the observed sequence of input tokens ($X_1 = x_1, \ldots, X_M = x_M$), a single forward pass through the language model gives an estimate of the *joint probability*,

$$p(X_1 = x_1) \cdot p(X_2 = x_2 \mid X_1 = x_1) \cdot \ldots \cdot p(X_M = x_M \mid X_1 = x_1, \ldots, X_{M-1} = x_{M-1}). \tag{D.1}$$

We highlight that Eq. (D.1) gives the joint probability *at the observed outcomes*: we obtain M "one step ahead" predictions, each conditioned only on observed outcomes and not on model predictions. Equation (D.1) is a common objective in LLM training, where "the joint probability the model assigns to the observations" is sometimes referred to as *teacher forcing* (Williams and Zipser, 1989).

At test time, LLMs are usually iteratively conditioned on their own *predictions*, generating novel outputs via multiple forward passes, i.e. one first samples $\hat{x}_M \sim p(X_M|\ldots)$, and then $\hat{x}_{M+1} \sim p(X_{M+1} \mid \ldots, X_M = \hat{x}_M)$, and so on. We here use "$\ldots$" to stand in for any additional tokens also conditioned on,

e.g. $(x_1, \ldots, x_{M-1})$. One usually ignores all other terms of the joint here – the predictions for $(X_1, \ldots, X_{M-1})$ that are generated in each forward pass – as only the last term $p(X_M \mid \ldots)$ is needed to sample the next token, i.e. the label in standard few-shot ICL applications.

Single-Forward Pass ICL Training Dynamics We now explain our approach for efficient evaluation of ICL training dynamics. Given input tokens $(X_1, \ldots, X_M)$ for the few-shot ICL setup described above, we first select those terms from Eq. (D.1) that correspond to label token predictions,

$$\prod_{i=1}^{N} p(X_{l_i} = x_{l_i} \mid X_1 = x_1, \ldots, X_{m<l_i} = x_{m<l_i}). \tag{D.2}$$

For each term, the model predicts a distribution over the entire token vocabulary, i.e. $p(X_{l_i} \mid \ldots)$ is a categorical distribution, $p = (p_1, \ldots, p_D)$, which is then evaluated at the observed tokens in Eq. (D.2). We can transform this into a prediction over only the few-shot task label Y by selecting the indices of the categorical distribution which correspond to the tokenized labels and then renormalizing, $p(Y) = (p_{t_1}, \ldots, p_{t_C} | \ldots)$, where C is the number of unique labels which are encoded to tokens $(t_1, \ldots, t_C)$. With this, we can rewrite Eq. (D.2) as the joint probability the model assigns to the sequence of labels given input sentences,

$$\begin{aligned} &p(Y_1 = y_1 \mid S_1 = s_1) \cdot p(Y_2 = y_2 \mid S_1 = s_1, Y_1 = y_1, S_2 = s_2) \cdot \ldots \\ &\cdot p(Y_N = y_N \mid S_1 = s_1, Y_1 = y_1, \ldots, S_{N-1} = s_{N-1}, Y_{N-1} = y_{N-1}, S_N = s_N). \end{aligned} \tag{D.3}$$

Note how, because the joint is evaluated at the observations, its individual terms are always conditioned on the true labels and not previous predictions. This allow us to cheaply compute the *training dynamics* of ICL as a function of increasing in-context dataset size. With each forward pass, we obtain the individual terms of Eq. (D.3), which are the few-shot ICL predictions at all possible context dataset sizes, $i = (1, \ldots, N)$.

In contrast, in standard few-shot ICL evaluations, each forward pass only yields predictions for a single test query, neglecting the information the joint contains about the first $N - 1$ label predictions.

There may be interesting applications of Eq. (D.3) to model selection, as the quantity has links to both Bayesian evidence (Murphy, 2022) and cross-validation (Fong and Holmes, 2020), although we do not explore this any further here.

Multi-Token Labels So far, we have assumed that each label string is encoded as a single token. However, our approach can also be applied if some or all labels are encoded as multiple tokens. In essence, we continue to measure only the probability the model assigns to the first token of each label, making the (fairly harmless) assumption that the first (or only) token that each label is encoded to is unique among labels. We believe this is justified, as, given the first token for a label, the model should near-deterministically predict the remaining tokens, i.e. all the predictive information is contained in the first token the model predicts for a label. For example, for the Subjectivity dataset, the label "objective" is encoded by the LLaMa tokenizer as a single token but the label "subjective" is encoded as two tokens, [subject, ive]. We only use the probability assigned to [subject] to assign probabilities to "subjective", and ignore any predictions for [ive]. However, if the model successfully accommodates the pattern of the in-context example labels, we would expect probabilities for [ive] following [subject] to be close to 1 always.

For LLaMa-7B on Subjectivity, we have investigated the above assumption empirically. After the first observation of the "subjectivity" label in-context, the probability of predicting [ive] after observing [subject] are 0.9998 ± 0.0003 for the following 12 instances of the "subjectivity" label, with probabilities normalized over *all* tokens of the vocabulary here. In other words, we can safely evaluate the performance of the LLaMa model from its predictions of only the [subject] token, even though the full label is split over two tokens [subject, ive].

D.3 Authorship Identification Task

We here give details on our novel authorship identification task.

Data Collection & Processing We extract the last 151 messages sent between two authors of the original paper on the Slack messaging platform. If multiple messages are sent in a row by one author, these count as multiple inputs. We filter out 42 messages that were of low quality: URLs, article summaries, missed call notifications, and meeting notes. This leaves us with 58 and 51 messages per author. We set the maximum message length to be 200 and truncate any messages longer than that. Before truncation, the longest message was 579 characters long. The median message length is 68 before and after truncation, mean and standard deviation shrink from 100 ± 98 to 88 ± 65. For use in ICL, we treat this dataset as we would any other and present messages in random order.

Data Release For now, we have decided to not make this dataset public for two reasons: (1) It contains genuinely private communication, and (2) releasing the data would mean that future LLMs might be trained on it, so we could no longer use it to test their ability to learn truly novel label relationships in-context. However, below, we give 6 random examples from the dataset:

Author 1 Would 10.30am on Tuesday work?

Author 1 Sounds good. When are you back again?

Author 1 Yeah, we might have to find somewhere else depending on whether my office mates are in, but its out of term so should be plenty of free meeting rooms if needed

Author 2 No problem!

Author 2 I'll be in [redacted] next week, so do you think we can meet in person?

Author 2 The vacation is 16 days!

D.4 Dataset Citations

We evaluate on SST-2 (Socher et al., 2013), Subjective (Wang and Manning, 2012), Financial Phrasebank (Malo et al., 2014), Hate Speech (Gibert et al., 2018), AG News (Zhang et al., 2015), Medical Questions Pairs (MQP) (McCreery et al., 2020),

as well as Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005), Recognizing Textual Entailment (RTE) (Dagan et al., 2005), and Winograd Schema Challenge (WNLI) (Levesque et al., 2012) from GLUE (Wang et al., 2019a).

D.5 Experiment Details

Below we give additional details on our experimental evaluation.

Guessing Baseline In our experiments, we frequently display a "guessing based on class frequencies" baseline as a grey-dashed line. This baseline presents an *informed guess* that relies only on knowing the class frequencies of the task and makes the exact same prediction for each input datapoint. We here explain how we compute this baseline for accuracy, entropy, and log likelihood. We are given a classification task with C classes which appear with frequencies $p = [p_1, \dots, p_C]$ in the training set. The baseline always predicts $p = [p_1, \dots, p_C]$, i.e. it predicts the class frequencies. For accuracy, it thus always predicts the majority class $c^* = \arg\max_k p_k$, which leads to accuracy p_{c^*}. Further, the baseline prediction leads to a log likelihood of $\sum_k p_k \log p_k$ and an entropy of $-\sum_k p_k \log p_k$ under the training data distribution.

Class Flipping While most of our tasks are *binary* classification, Financial Phrasebank and AG News are not. For these datasets, when "flipping" labels in Section 6.7 and Section 6.8, we actually rotate labels instead, i.e. we reassign labels as $y \leftarrow (y + 1) \mod C$, where C is the number of classes. For AG News, ['world', 'sports', 'business', 'science and technology'] get mapped to ['sports', 'business', 'science and technology', 'world']. For Financial Phrasebank, ['negative', 'neutral', 'positive'] get mapped to ['neutral', 'positive', 'negative']. Note that, for Financial Phrasebank, rotating the labels is harder than naively inverting the label order, as rotating does not leave the meaning of the "neutral" label unchanged.

In-Context Example Formatting We use the following simple templates to format the in-context examples. For SST-2, Subjectivity, Financial Phrasebank, Hate Speech, and our author identification task, we use the following line of Python code to format each input example:

```
f"Sentence: '{sentence}'\nAnswer: {label}\n\n".
```

For MRPC, WNLI, and RTE, we format instances with

```
f"Sentence 1: '{sentence1}'\nSentence 2: '{sentence2}'\nAnswer: {label}\n\n".
```

For MQP, we use

```
f"Question 1: '{sentence1}'\nQuestion 2: '{sentence2}'\nAnswer: {label}\n\n".
```

Implementation We rely on the Hugging Face Python library (Wolf et al., 2020) and PyTorch (Paszke et al., 2019) to implement the experiments of this chapter. We use half-precision floating-point numbers for LLaMa-65B and LLaMa-2-70B, and we use 8 bit-quantization for all other models, which we have found to not affect performance notably. In Fig. D.2, we illustrate this by showing the difference between 8 bit quantization and full 32 bit precision for default ICL and ICL with label randomization for LLaMa-2-7B on the Subjectivity dataset: there is no significant loss of precision or change in behavior from 8 bit quantization.

Datasets We use Hugging Face Spaces to access all tasks considered in this chapter. For Hate Speech, we select the first 1000 examples with labels 0 and 1, skipping datapoints with labels 2 and 3. We do not use custom processing for any other dataset.

Whitespace Tokenization To evaluate few-shot ICL performance as introduced in Section 6.4, we need to identify the tokens that individual task labels are encoded to. We here detail how to achieve this at the example of the SST-2 label "positive". In particular, we highlight the, perhaps unexpected, effects of whitespaces on input tokenization. These details are important and, if not considered correctly, can degrade performance significantly.

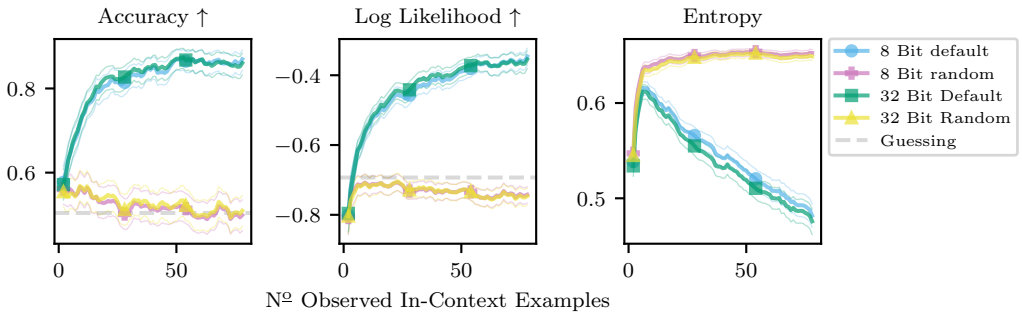


Figure D.2: Few-shot ICL at 8 bit and 32 bit precision with default and randomized labels for LLaMa-2-7B on Subjectivity. There is no significant performance degradation from 8 bit quantization. We average over 500 random in-context datasets, thin lines are bootstrapped 99 % confidence intervals, and we apply moving averages (window size 5) for clarity.

For Falcon models, the tokenizer encodes "`Answer:`" as `[20309, 37]`, "`Answer:-`" as `[20309, 37, 204]`, and "`Answer:-positive`" as `[20309, 37, 3508]`. We here use dashes "`-`" instead of whitespaces for improved legibility. Clearly, the relevant token for the label "positive" is `[3508]`. Note how the token `[204]` for the trailing whitespace disappears again after appending the label. Further, just encoding "`positive`" without a preceding whitespace gives `[28265]`. However, this token does not appear in the input, where the label is preceded by a whitespace – we should thus use the token `[3508]` to measure ICL performance

The LLaMa and LLaMa-2 tokenizer encodes "`Answer:`" as `[673, 29901]`, "`Answer:-`" as `[673, 29901, 29871]`, and "`Answer:-positive`" as `[673, 29901, 6374]`. Thus, the relevant token for the label "positive" is `[6374]`. In contrast to the Falcon tokenizer, just encoding "`positive`" without a preceding whitespace also gives `[6374]`.

Lastly, similar caveats apply to the classic evaluation procedure for few-shot ICL, where we only evaluate the prediction for a single test query at the end of the input. Here, it is crucially important that we do not end inputs with a trailing whitespace. As we have seen above, for both LLaMa and Falcon tokenizers, the trailing whitespace leads to the generation of an extra token that is not present when encoding complete in-context examples, as the whitespace would usually be included in the label prediction itself. This change in tokenization between in-context examples and test query can adversely affect ICL performance.

Table D.1: Maximum number of in-context examples we consider for each model-task combination. Below, we shorten AG News as AGN, Hate Speech as HS, and Financial Phrasebank as FP.

	SST-2	Subj	FP	HS	AGN	MQP	MRPC	RTE	WNLI
LLaMa-2	140	79	73	76	45	47	40	28	57
LLaMa	66	37	33	26	21	21	20	13	27
Falcon	67	39	37	28	25	23	21	15	30

Statistical Significance In Tables 6.1, 6.2, D.2 and D.3 we bold differences if they are statistically significant at a 95 % level. Concretely, we compute if the absolute average differences are larger than 1.96 times the standard error. Similarly, when deciding if default label performance is significantly better than random guessing performance, we check if mean performance plus 1.645 times the standard error is larger than the guessing baseline across accuracy and log likelihood.

Maximum Context Dataset Size For each task, we create in-context datasets by sub-sampling from the training set of the task. Falcon and LLaMa support input sizes up to 2048 tokens, and LLaMa-2 supports up to 4096 input tokens. For all models, performance will degrade if the input size exceeds this limit. This caps the number of in-context examples we can include for each task. For tasks where the individual input sentences are longer, we will be able to include fewer examples in-context. Across all in-context datasets that we sample for a task, we compute the minimum number of in-context examples needed to exceed the token limit of 2048 or 4096. This is the maximum in-context dataset size up to which we report results for that task. We list these numbers in Table D.1

Calibration We do not calibrate predicted probabilities by first dividing them by a "prior" probability and then renormalizing as suggested by Zhao et al. (2021). We have found this rarely improves, and sometimes degrades predictions, cf. Fig. D.3. We observe this happening in particular for tasks where labels are encoded as multiple tokens.

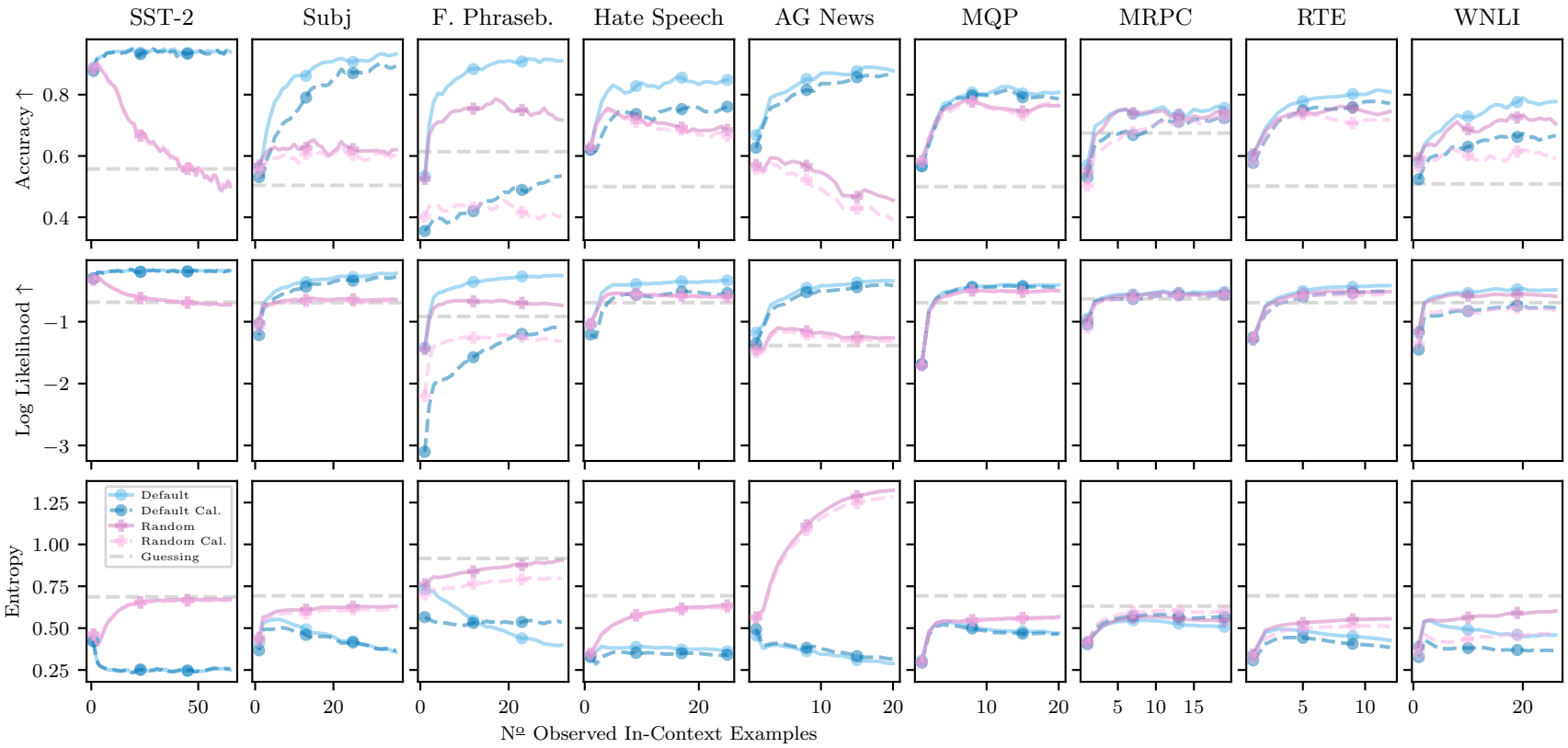


Figure D.3: Few-shot ICL with randomized labels for **LLaMa-65B**. We additionally display performance for random and default labels when "calibrating" the predicted probabilities as suggested by Zhao et al. (2021). We do not find calibration helpful to improve ICL performance. We average over 500 random in-context datasets and thin lines are bootstrapped 99 % confidence intervals and we apply a moving average (window size 3) for clarity.

D.6 Additional Results

Section 6.4 – Training Dynamics Fig. D.4 shows few-shot ICL training dynamics on SST-2 for a selection of models at different parameter counts.

Section 6.5 – Label Randomization Table D.2 gives full summary statistics across all models, tasks, and metrics for the label randomization experiment. Due to space limitations, we refer the reader to the original publication for full training curves across models and tasks.

Section 6.6 – Author ID Task Fig. D.6 gives few-shot ICL results for all models on our novel authorship identification task.

Section 6.7 – Flipped Labels Table D.3 gives full summary statistics across all models, tasks, and metrics for the difference between default and flipped label performance. Due to space limitations, we refer the reader to the original publication for full training curves across models, tasks, and additional replacement labels.

Section 6.8 – Dynamic Label Flipping In Figs. D.7 to D.11, we give results for the experiments investigating NH3 for all large models on tasks where label flipping gave strong performance in Section 6.7. For LLaMa-2-70B on Hate Speech in Fig. D.10, metrics appear very similar initially. However, when exploring additional changepoints in Fig. D.12, we do find significant differences in predictions.

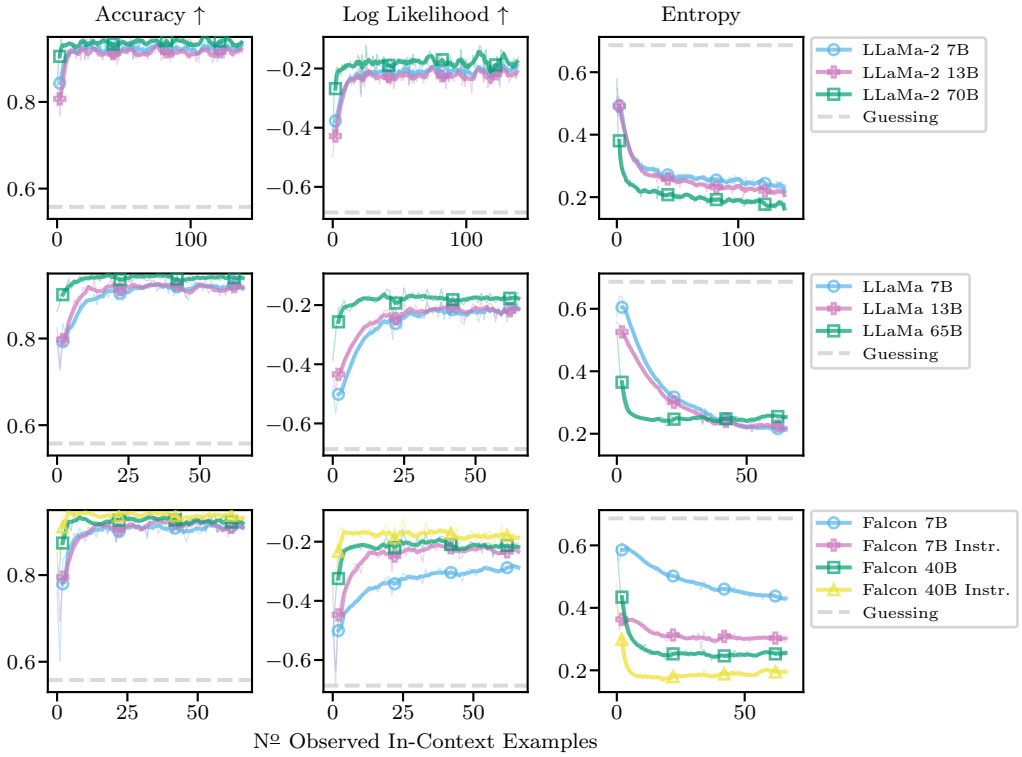


Figure D.4: Few-shot ICL **training dynamics** in a standard scenario on **SST-2**. Accuracy (↑) and log likelihood (↑) improve with in-context dataset size, and entropies decrease appropriately. Averages over 500 random subsets of the SST-2 training set (thin lines), additionally applying a moving average with window size 5 for clarity (thick lines).

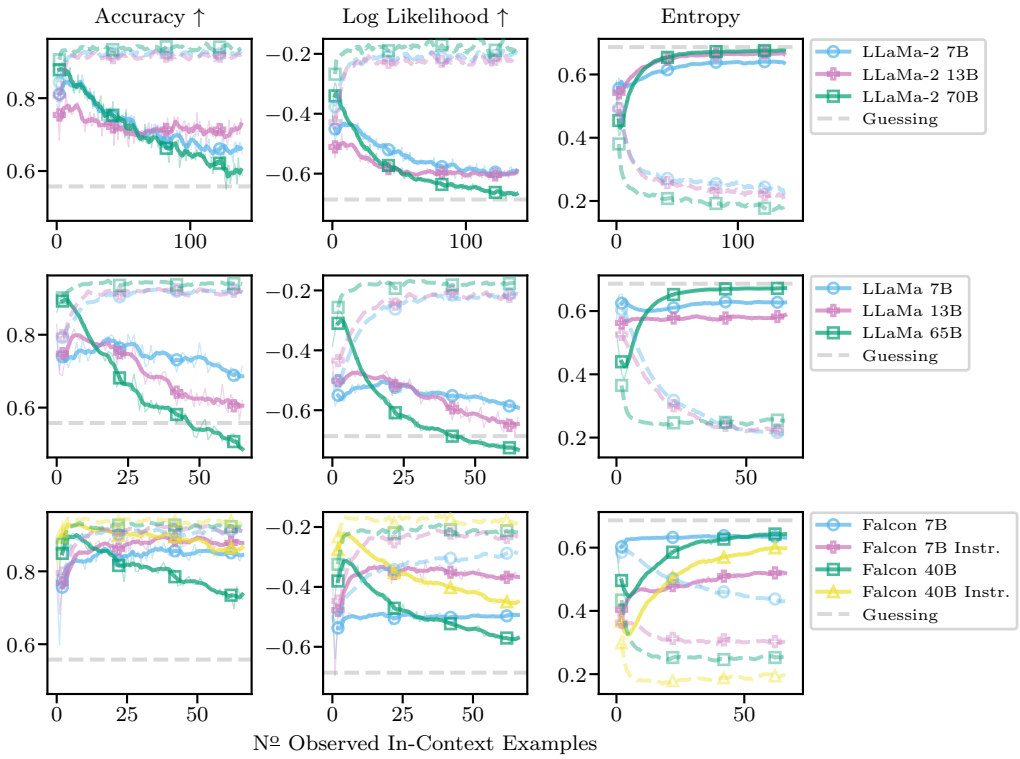


Figure D.5: Few-shot ICL with **randomized labels** for **SST-2**: Compared to default ICL behavior (dashed lines, cf. Fig. D.4), log likelihoods and entropies of the models degrade when in-context example labels are randomized. Thin lines are averages over 500 repetitions, thick lines with moving average (window size 5), guessing baseline based on class frequencies.

Table D.2: Full summary statistics for the randomization experiment of Section 6.5. We strongly encourage readers to also view the full training curves across all possible context sizes in the original publication. We here show the average difference between the default and randomized label scenario across all metrics. We compute "Metric(Default) - Metric(Random)", such that positive accuracy/log likelihood differences indicate that ICL performs worse with randomized labels. We compute differences at the maximum context size for each task-model combination (cf. Appendix D.5 and Table D.1). We display experiments where ICL accuracies or log likelihoods on the default labels do not significantly exceed random guessing performance in lightgray. Across models, tasks, and metrics, model performance is usually significantly worse when labels are randomized and default performance exceeds random guessing. We display mean differences and standard errors over 500 runs. We bold entries for which mean differences are statistically significant.

Δ Log Lik.	SST-2	Subj	F. Phraseb.	Hate Speech	AG News	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**0.42 ± 0.02**	**0.39 ± 0.02**	**0.57 ± 0.02**	**0.18 ± 0.01**	**0.53 ± 0.04**	**0.03 ± 0.01**	0.02 ± 0.01	**0.03 ± 0.01**	0.02 ± 0.01
LLaMa-2 13B	**0.41 ± 0.02**	**0.62 ± 0.03**	**0.49 ± 0.03**	**0.24 ± 0.02**	**0.81 ± 0.04**	**0.04 ± 0.01**	0.01 ± 0.01	**0.06 ± 0.02**	0.02 ± 0.03
LLaMa-2 70B	**0.51 ± 0.03**	**0.53 ± 0.02**	**0.57 ± 0.02**	**0.34 ± 0.02**	**0.80 ± 0.03**	**0.29 ± 0.02**	**0.04 ± 0.02**	**0.22 ± 0.02**	**0.18 ± 0.02**
LLaMa 7B	**0.39 ± 0.03**	**0.42 ± 0.03**	**0.36 ± 0.02**	**0.15 ± 0.02**	**0.30 ± 0.03**	**0.03 ± 0.01**	0.00 ± 0.01	0.03 ± 0.02	0.01 ± 0.02
LLaMa 13B	**0.44 ± 0.03**	**0.45 ± 0.02**	**0.37 ± 0.02**	**0.16 ± 0.02**	**0.32 ± 0.03**	**0.04 ± 0.02**	−0.01 ± 0.02	**0.05 ± 0.02**	0.02 ± 0.02
LLaMa 65B	**0.55 ± 0.02**	**0.45 ± 0.02**	**0.49 ± 0.02**	**0.23 ± 0.02**	**0.88 ± 0.04**	**0.14 ± 0.02**	0.01 ± 0.01	**0.12 ± 0.02**	**0.08 ± 0.02**
Falcon 7B	**0.20 ± 0.01**	**0.19 ± 0.01**	**0.25 ± 0.01**	**0.06 ± 0.01**	**0.31 ± 0.03**	0.01 ± 0.02	0.01 ± 0.02	−0.01 ± 0.02	0.01 ± 0.02
Falcon 7B Instr.	**0.13 ± 0.01**	**0.08 ± 0.01**	**0.11 ± 0.01**	**0.03 ± 0.02**	**0.15 ± 0.02**	0.03 ± 0.02	0.02 ± 0.03	−0.00 ± 0.02	0.00 ± 0.02
Falcon 40B	**0.34 ± 0.02**	**0.35 ± 0.02**	**0.31 ± 0.02**	**0.18 ± 0.02**	**0.90 ± 0.04**	**0.06 ± 0.02**	0.01 ± 0.02	0.01 ± 0.02	0.02 ± 0.02
Falcon 40B Instr.	**0.25 ± 0.02**	**0.37 ± 0.03**	**0.27 ± 0.02**	0.02 ± 0.03	**0.77 ± 0.04**	**0.06 ± 0.02**	0.02 ± 0.02	0.02 ± 0.02	**0.04 ± 0.02**

Δ Accuracy	SST-2	Subj	F. Phraseb.	Hate Speech	AG News	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**28.4 ± 2.1**	**39.8 ± 2.4**	**30.4 ± 2.4**	**16.2 ± 2.3**	**13.4 ± 2.2**	3.8 ± 2.2	5.2 ± 2.8	4.0 ± 2.4	3.0 ± 2.5
LLaMa-2 13B	**19.8 ± 2.0**	**46.0 ± 2.4**	**24.4 ± 2.4**	**19.0 ± 2.3**	**34.6 ± 2.3**	3.2 ± 2.5	**2.8 ± 1.4**	1.6 ± 1.9	3.6 ± 2.3
LLaMa-2 70B	**34.8 ± 2.3**	**29.8 ± 2.1**	**27.0 ± 2.2**	**28.6 ± 2.5**	**24.4 ± 2.1**	**19.6 ± 2.5**	**3.6 ± 1.7**	**14.0 ± 2.0**	**13.6 ± 2.5**
LLaMa 7B	**24.6 ± 2.2**	**36.4 ± 2.5**	**19.8 ± 2.1**	**10.8 ± 2.2**	**11.2 ± 2.0**	4.6 ± 2.8	−0.2 ± 1.9	2.8 ± 2.6	1.0 ± 2.0
LLaMa 13B	**31.4 ± 2.2**	**42.4 ± 2.5**	**26.4 ± 2.7**	**13.8 ± 2.0**	**9.2 ± 1.8**	4.0 ± 2.6	−2.6 ± 2.4	2.0 ± 2.4	3.0 ± 2.5
LLaMa 65B	**47.4 ± 2.5**	**34.4 ± 2.3**	**18.4 ± 2.1**	**13.6 ± 2.2**	**39.2 ± 2.7**	**8.4 ± 2.0**	2.0 ± 1.4	**7.6 ± 1.8**	**4.6 ± 2.1**
Falcon 7B	**6.8 ± 1.6**	**14.4 ± 2.0**	**24.0 ± 2.4**	**7.6 ± 1.8**	**13.2 ± 2.0**	3.0 ± 2.2	2.6 ± 2.4	−1.4 ± 2.9	0.0 ± 2.1
Falcon 7B Instr.	**3.6 ± 1.4**	**7.6 ± 1.8**	**5.2 ± 1.5**	2.0 ± 2.0	**8.2 ± 1.9**	1.6 ± 2.6	3.0 ± 2.9	−1.0 ± 2.3	2.8 ± 1.8
Falcon 40B	**20.2 ± 2.1**	**22.2 ± 2.1**	**10.0 ± 1.8**	**10.6 ± 2.2**	**49.4 ± 2.4**	**11.2 ± 2.4**	−0.4 ± 1.1	0.4 ± 1.7	1.2 ± 2.2
Falcon 40B Instr.	**6.8 ± 1.4**	**21.8 ± 2.2**	**10.4 ± 1.8**	0.8 ± 1.6	**40.4 ± 2.4**	**4.2 ± 2.0**	3.4 ± 2.3	3.2 ± 1.9	3.6 ± 2.2

Δ Entropy	SST-2	Subj	F. Phraseb.	Hate Speech	AG News	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**−0.465 ± 0.007**	**−0.177 ± 0.008**	**−0.529 ± 0.012**	**−0.099 ± 0.006**	**−0.910 ± 0.014**	**−0.004 ± 0.002**	**−0.005 ± 0.002**	**−0.014 ± 0.004**	0.004 ± 0.003
LLaMa-2 13B	**−0.499 ± 0.008**	**−0.429 ± 0.009**	**−0.477 ± 0.013**	**−0.204 ± 0.008**	**−1.009 ± 0.013**	**−0.018 ± 0.004**	−0.021 ± 0.004	**−0.079 ± 0.006**	**−0.058 ± 0.006**
LLaMa-2 70B	**−0.564 ± 0.007**	**−0.500 ± 0.007**	**−0.563 ± 0.010**	**−0.313 ± 0.010**	**−1.046 ± 0.011**	**−0.242 ± 0.009**	**−0.074 ± 0.006**	**−0.246 ± 0.009**	**−0.220 ± 0.008**
LLaMa 7B	**−0.426 ± 0.009**	**−0.153 ± 0.009**	**−0.225 ± 0.011**	**−0.103 ± 0.007**	**−0.497 ± 0.013**	−0.002 ± 0.001	−0.001 ± 0.003	−0.002 ± 0.004	−0.001 ± 0.004
LLaMa 13B	**−0.376 ± 0.009**	**−0.193 ± 0.009**	**−0.218 ± 0.009**	**−0.112 ± 0.007**	**−0.567 ± 0.015**	−0.008 ± 0.004	**−0.014 ± 0.005**	**−0.012 ± 0.004**	0.001 ± 0.004
LLaMa 65B	**−0.422 ± 0.009**	**−0.283 ± 0.008**	**−0.502 ± 0.011**	**−0.273 ± 0.009**	**−1.032 ± 0.013**	**−0.100 ± 0.007**	**−0.040 ± 0.005**	**−0.128 ± 0.008**	**−0.153 ± 0.008**
Falcon 7B	**−0.196 ± 0.007**	**−0.109 ± 0.006**	**−0.099 ± 0.006**	**−0.046 ± 0.005**	**−0.428 ± 0.015**	−0.000 ± 0.004	0.003 ± 0.005	−0.002 ± 0.004	−0.002 ± 0.004
Falcon 7B Instr.	**−0.207 ± 0.007**	**−0.038 ± 0.004**	**−0.115 ± 0.007**	**−0.041 ± 0.006**	**−0.200 ± 0.011**	0.000 ± 0.004	0.007 ± 0.006	0.003 ± 0.006	−0.003 ± 0.006
Falcon 40B	**−0.382 ± 0.009**	**−0.248 ± 0.009**	**−0.358 ± 0.010**	**−0.175 ± 0.008**	**−0.923 ± 0.015**	−0.002 ± 0.004	0.001 ± 0.006	−0.008 ± 0.005	−0.001 ± 0.004
Falcon 40B Instr.	**−0.396 ± 0.008**	**−0.177 ± 0.009**	**−0.313 ± 0.010**	**−0.202 ± 0.009**	**−0.922 ± 0.015**	**−0.039 ± 0.006**	−0.013 ± 0.007	**−0.033 ± 0.006**	**−0.012 ± 0.005**

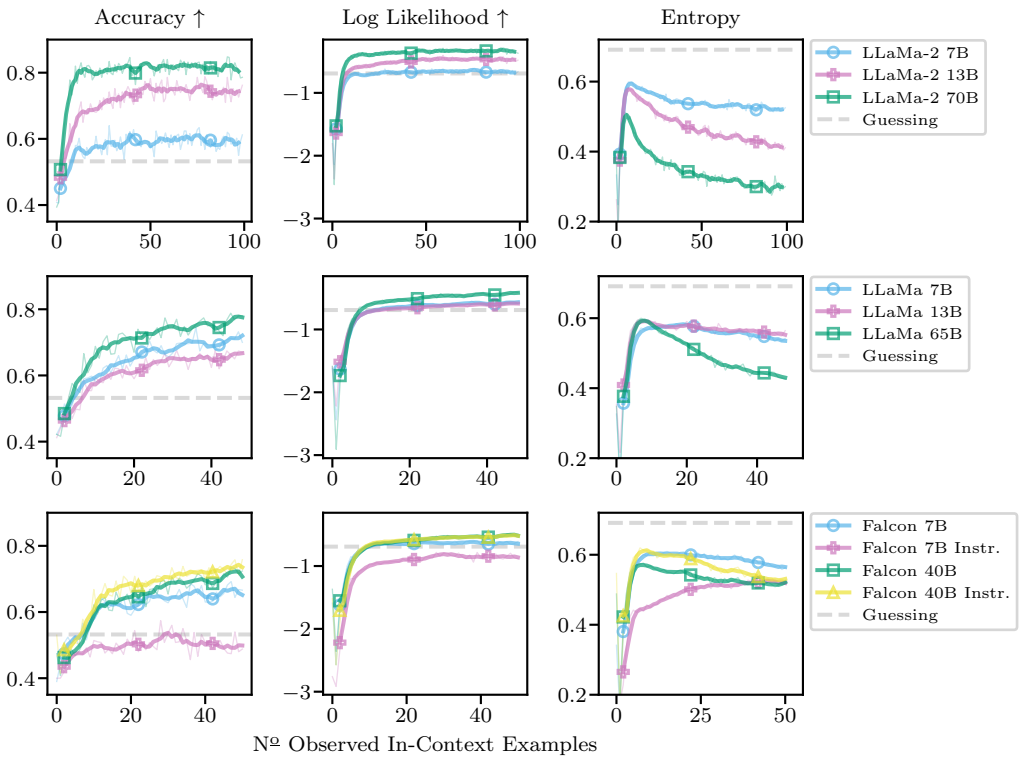


Figure D.6: Few-shot ICL on our **novel author identification** task for all models. Models achieve accuracies significantly better than random guessing on the author identification task. Thus, ICL predictions must depend on the label relationship provided in-context. Thin lines are averages over 500 runs, thick lines with moving average (window size 5).

Table D.3: Full summary statistics for the flipped label experiments of Section 6.7. We strongly encourage readers to also view the full training curves across all possible context sizes and additional replacement labels in the original publication. We here show the average difference between the default and flipped label scenario. We compute "Metric(Default) - Metric(Flipped)", such that positive accuracy/log likelihood differences indicate that ICL performs worse with flipped labels. We compute differences at the maximum context size for each task-model combination (cf. Appendix D.5 and Table D.1). We display experiments where ICL accuracies or log likelihoods on the default labels do not significantly exceed random guessing performance in lightgray. Across models, tasks, and metrics, model performance is usually significantly worse when labels are flipped (and default performance exceeds random guessing). In particular for entropies, we observe significant differences between ICL predictions. We display mean differences and standard errors over 100 runs. We bold entries for which mean differences are statistically significant.

Δ Log Lik.	SST-2	Subj	F. Phraseb.	Hate Speech	AG News	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**0.41 ± 0.02**	−0.04 ± 0.02	**0.36 ± 0.03**	**0.26 ± 0.01**	**0.87 ± 0.03**	0.01 ± 0.00	−0.02 ± 0.01	**0.22 ± 0.01**	−0.00 ± 0.01
LLaMa-2 13B	**0.31 ± 0.02**	0.03 ± 0.02	**0.30 ± 0.02**	**0.26 ± 0.02**	**0.90 ± 0.03**	**0.15 ± 0.01**	**0.15 ± 0.01**	**0.38 ± 0.02**	**0.20 ± 0.02**
LLaMa-2 70B	**0.08 ± 0.02**	**0.08 ± 0.02**	**0.35 ± 0.03**	**0.18 ± 0.02**	**0.90 ± 0.02**	**0.24 ± 0.02**	**0.15 ± 0.02**	**0.21 ± 0.02**	0.08 ± 0.02
LLaMa 7B	**0.36 ± 0.02**	**0.13 ± 0.02**	**0.53 ± 0.02**	**0.31 ± 0.02**	**1.06 ± 0.03**	0.05 ± 0.00	**0.06 ± 0.01**	**0.14 ± 0.01**	−0.03 ± 0.01
LLaMa 13B	**0.40 ± 0.02**	**0.09 ± 0.02**	**0.51 ± 0.02**	**0.33 ± 0.02**	**1.18 ± 0.03**	**0.16 ± 0.01**	**0.16 ± 0.01**	**0.22 ± 0.01**	0.05 ± 0.01
LLaMa 65B	**0.33 ± 0.02**	**0.16 ± 0.02**	**0.63 ± 0.02**	**0.19 ± 0.02**	**1.02 ± 0.03**	**0.39 ± 0.02**	**0.31 ± 0.02**	**0.41 ± 0.02**	**0.20 ± 0.02**
Falcon 7B	**0.48 ± 0.02**	**0.37 ± 0.01**	**0.58 ± 0.02**	**0.19 ± 0.01**	**1.38 ± 0.04**	0.05 ± 0.01	0.02 ± 0.01	0.05 ± 0.01	0.12 ± 0.01
Falcon 7B Instr.	**0.70 ± 0.02**	**0.48 ± 0.01**	**0.77 ± 0.02**	**0.52 ± 0.02**	**2.29 ± 0.03**	−0.07 ± 0.01	0.01 ± 0.02	0.15 ± 0.02	**0.22 ± 0.02**
Falcon 40B	**0.33 ± 0.02**	**0.25 ± 0.02**	**0.74 ± 0.03**	**0.27 ± 0.02**	**1.00 ± 0.03**	**0.20 ± 0.01**	**0.19 ± 0.01**	**0.39 ± 0.01**	0.09 ± 0.01
Falcon 40B Instr.	**0.45 ± 0.02**	**0.37 ± 0.02**	**0.94 ± 0.03**	**0.68 ± 0.02**	**0.89 ± 0.03**	**0.48 ± 0.02**	**0.15 ± 0.01**	**0.70 ± 0.02**	**0.19 ± 0.01**

Δ Accuracy	SST-2	Subj	F. Phraseb.	Hate Speech	AG News	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**34.0 ± 5.3**	−3.0 ± 4.3	**15.0 ± 4.3**	**26.0 ± 6.9**	**47.0 ± 5.6**	1.0 ± 8.4	−5.0 ± 5.9	**32.0 ± 7.5**	1.0 ± 7.3
LLaMa-2 13B	**14.0 ± 4.0**	1.0 ± 1.0	**12.0 ± 3.8**	**18.0 ± 5.4**	**37.0 ± 5.8**	**22.1 ± 7.1**	**20.0 ± 5.3**	**36.0 ± 7.0**	**27.0 ± 6.5**
LLaMa-2 70B	2.0 ± 1.4	1.0 ± 2.2	**13.0 ± 3.4**	0.0 ± 3.2	**51.0 ± 5.6**	**11.0 ± 4.7**	**20.0 ± 5.5**	7.0 ± 5.0	5.0 ± 3.8
LLaMa 7B	**22.0 ± 4.8**	3.0 ± 4.8	**36.0 ± 5.9**	**27.0 ± 6.6**	**64.0 ± 6.4**	6.0 ± 6.6	**13.0 ± 6.6**	**19.0 ± 7.0**	0.0 ± 8.6
LLaMa 13B	**26.0 ± 5.8**	**7.0 ± 3.5**	**26.0 ± 5.8**	**36.0 ± 5.9**	**55.0 ± 6.2**	**16.0 ± 6.7**	**12.0 ± 4.5**	**28.0 ± 7.5**	6.0 ± 6.9
LLaMa 65B	**15.0 ± 4.3**	**8.0 ± 3.7**	**27.0 ± 5.3**	**14.0 ± 5.8**	**50.0 ± 5.7**	**29.0 ± 7.4**	**41.0 ± 6.3**	**46.0 ± 6.7**	**30.0 ± 6.1**
Falcon 7B	**49.0 ± 5.9**	**39.0 ± 4.9**	**34.0 ± 6.7**	**20.0 ± 8.4**	**63.0 ± 6.7**	13.0 ± 8.2	7.0 ± 4.5	10.0 ± 6.1	12.0 ± 8.3
Falcon 7B Instr.	**63.0 ± 5.8**	**46.0 ± 7.0**	**48.0 ± 6.6**	**42.0 ± 7.4**	**71.0 ± 5.2**	−10.0 ± 7.1	−9.0 ± 6.3	11.0 ± 7.6	**17.0 ± 8.5**
Falcon 40B	**15.0 ± 4.3**	**17.0 ± 4.5**	**36.0 ± 5.9**	**20.0 ± 6.8**	**58.0 ± 6.5**	**28.0 ± 6.8**	**32.0 ± 6.8**	**25.0 ± 8.3**	**16.0 ± 7.8**
Falcon 40B Instr.	**29.0 ± 5.3**	**30.0 ± 5.2**	**45.0 ± 5.5**	**53.0 ± 6.8**	**49.0 ± 6.9**	**45.0 ± 7.0**	11.0 ± 6.3	**48.0 ± 7.0**	**22.0 ± 7.6**

Δ Entropy	SST-2	Subj	F. Phraseb.	Hate Speech	AG News	MQP	MRPC	RTE	WNLI
LLaMa-2 7B	**−0.519 ± 0.022**	0.010 ± 0.017	**−0.397 ± 0.026**	**−0.100 ± 0.013**	**−0.749 ± 0.030**	−0.003 ± 0.005	**0.049 ± 0.005**	**−0.027 ± 0.008**	−0.006 ± 0.008
LLaMa-2 13B	**−0.479 ± 0.019**	**−0.058 ± 0.015**	**−0.475 ± 0.024**	**−0.194 ± 0.019**	**−0.953 ± 0.028**	**−0.027 ± 0.010**	**−0.067 ± 0.011**	**−0.106 ± 0.018**	**−0.073 ± 0.017**
LLaMa-2 70B	**−0.167 ± 0.019**	**−0.100 ± 0.015**	**−0.398 ± 0.029**	**−0.255 ± 0.018**	**−0.995 ± 0.024**	**−0.243 ± 0.019**	**−0.150 ± 0.015**	**−0.258 ± 0.019**	**−0.209 ± 0.017**
LLaMa 7B	**−0.434 ± 0.020**	−0.017 ± 0.023	**−0.355 ± 0.023**	**−0.159 ± 0.019**	**−0.645 ± 0.033**	0.007 ± 0.003	**−0.032 ± 0.005**	−0.005 ± 0.008	−0.014 ± 0.010
LLaMa 13B	**−0.424 ± 0.023**	−0.005 ± 0.020	**−0.237 ± 0.022**	**−0.151 ± 0.018**	**−0.731 ± 0.033**	0.008 ± 0.009	0.019 ± 0.012	−0.010 ± 0.008	−0.009 ± 0.009
LLaMa 65B	**−0.345 ± 0.019**	**−0.140 ± 0.020**	**−0.528 ± 0.024**	**−0.273 ± 0.020**	**−1.041 ± 0.029**	**−0.128 ± 0.019**	**−0.141 ± 0.015**	**−0.205 ± 0.020**	**−0.199 ± 0.019**
Falcon 7B	**−0.276 ± 0.019**	**−0.118 ± 0.014**	−0.022 ± 0.018	**−0.062 ± 0.013**	**−0.518 ± 0.037**	0.002 ± 0.009	0.005 ± 0.006	−0.004 ± 0.010	−0.015 ± 0.013
Falcon 7B Instr.	**−0.367 ± 0.020**	**−0.068 ± 0.012**	**−0.327 ± 0.021**	**−0.052 ± 0.016**	**−0.220 ± 0.028**	−0.024 ± 0.009	**0.127 ± 0.015**	0.005 ± 0.015	0.001 ± 0.020
Falcon 40B	**−0.392 ± 0.020**	**−0.233 ± 0.022**	**−0.420 ± 0.027**	**−0.190 ± 0.022**	**−0.904 ± 0.033**	−0.003 ± 0.010	**−0.103 ± 0.012**	−0.018 ± 0.014	−0.001 ± 0.013
Falcon 40B Instr.	**−0.483 ± 0.020**	**−0.160 ± 0.024**	**−0.430 ± 0.028**	**−0.313 ± 0.024**	**−0.916 ± 0.033**	**−0.098 ± 0.017**	−0.017 ± 0.014	**−0.063 ± 0.017**	−0.013 ± 0.013

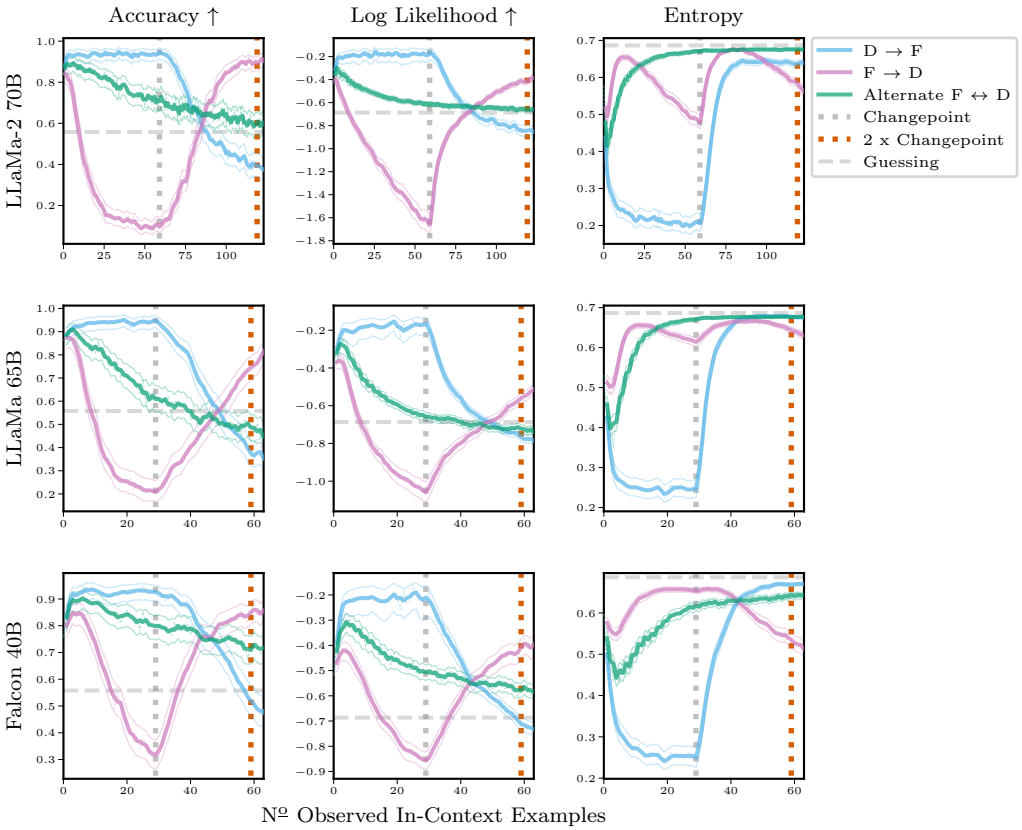


Figure D.7: Few-shot ICL on **SST-2** when the **label relationship changes throughout ICL**. For (D → F), we start with default labels and change to flipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point – but they are not. Averages over 500 repetitions, we apply moving averages (window size 3), and thin lines are bootstrapped 99 % confidence intervals, thin dashed lines are mean results without moving average smoothing.

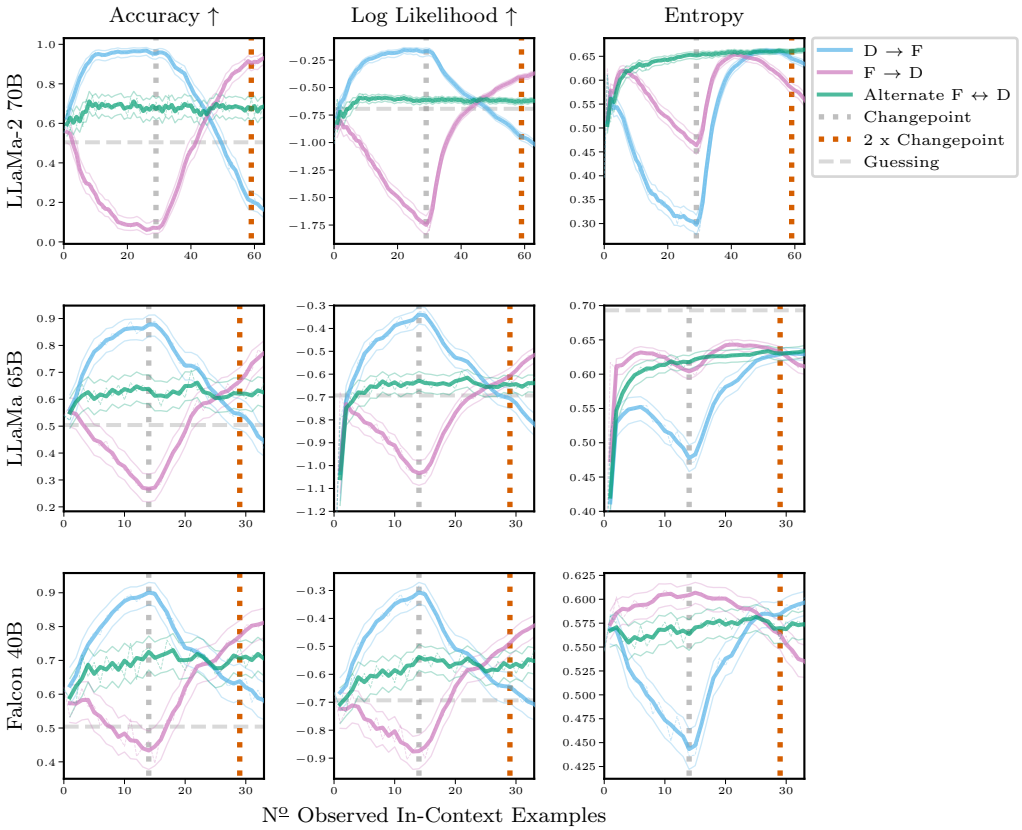


Figure D.8: Few-shot ICL on **Subjectivity** when the **label relationship changes throughout ICL**. For (D → F), we start with default labels and change to flipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point – but they are not. Averages over 500 repetitions, we apply moving averages (window size 3), and thin lines are bootstrapped 99 % confidence intervals, thin dashed lines are mean results without moving average smoothing.

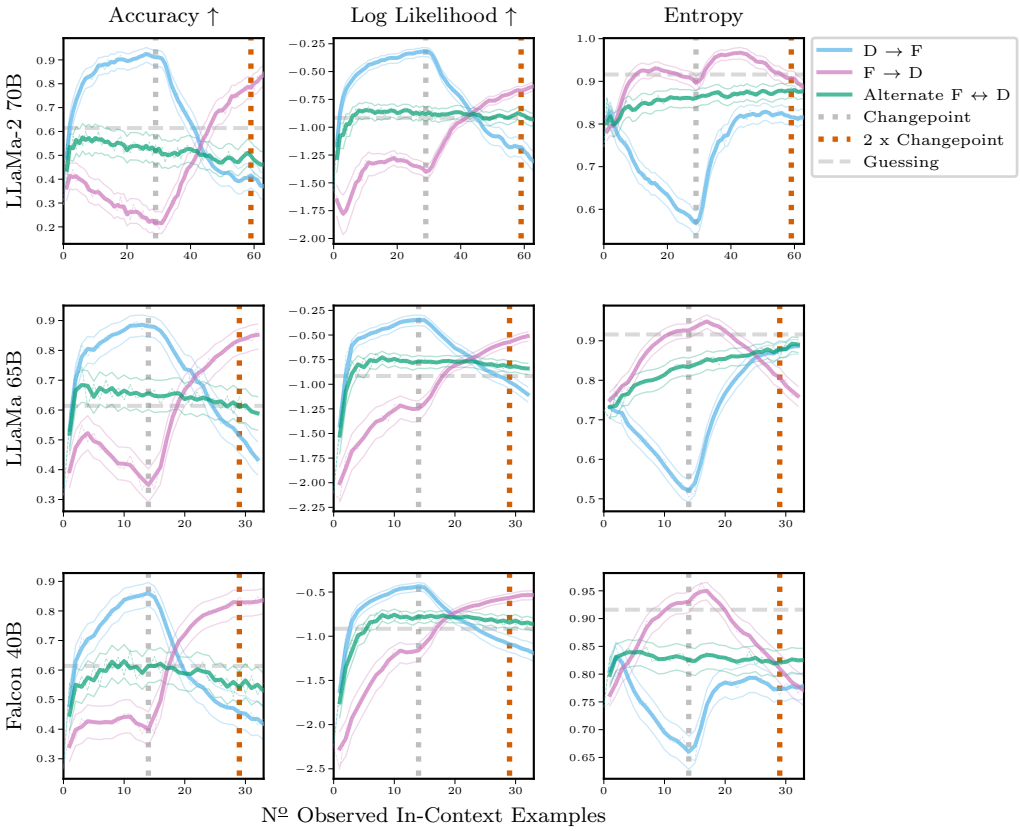


Figure D.9: Few-shot ICL on **Financial Phrasebank** when the **label relationship changes throughout ICL**. For (D → F), we start with default labels and change to flipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point – but they are not. Averages over 500 repetitions, we apply moving averages (window size 3), and thin lines are bootstrapped 99 % confidence intervals, thin dashed lines are mean results without moving average smoothing.

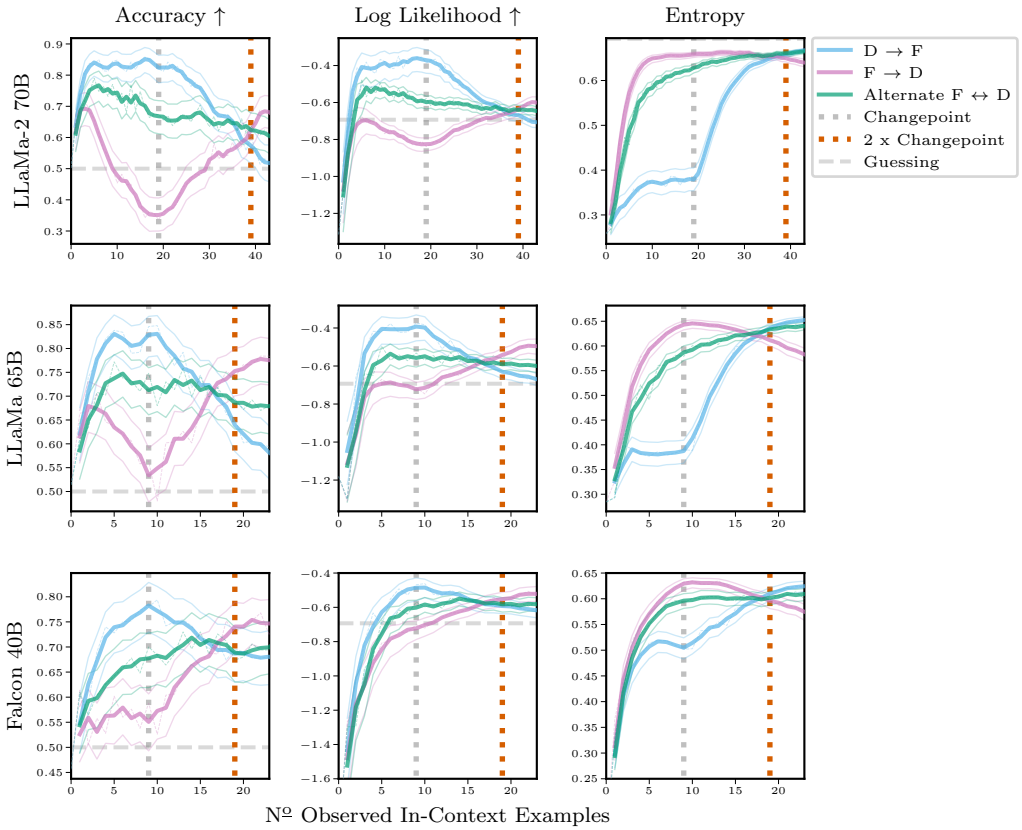


Figure D.10: Few-shot ICL on **Hate Speech** when the **label relationship changes throughout ICL**. For (D → F), we start with default labels and change to flipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point – but they are not. Averages over 500 repetitions, we apply moving averages (window size 3), and thin lines are bootstrapped 99 % confidence intervals, thin dashed lines are mean results without moving average smoothing.

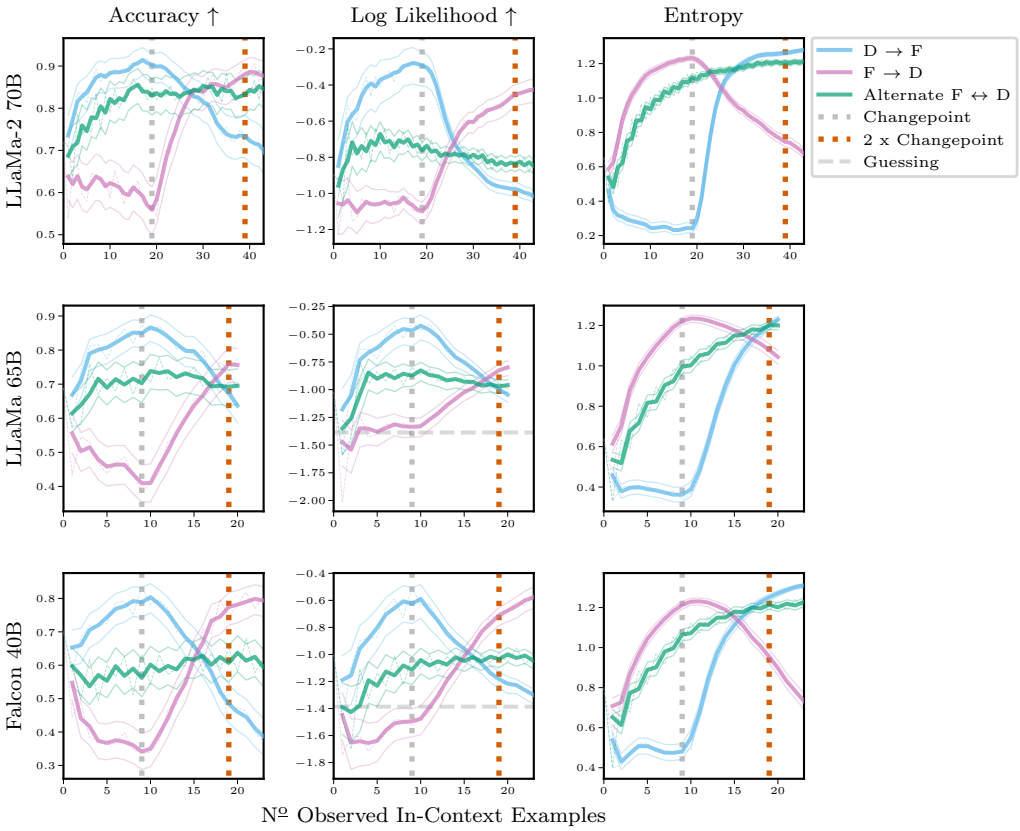


Figure D.11: Few-shot ICL on **AG News** when the **label relationship changes throughout ICL**. For (D → F), we start with default labels and change to flipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point – but they are not. Averages over 500 repetitions, we apply moving averages (window size 3), and thin lines are bootstrapped 99 % confidence intervals, thin dashed lines are mean results without moving average smoothing.

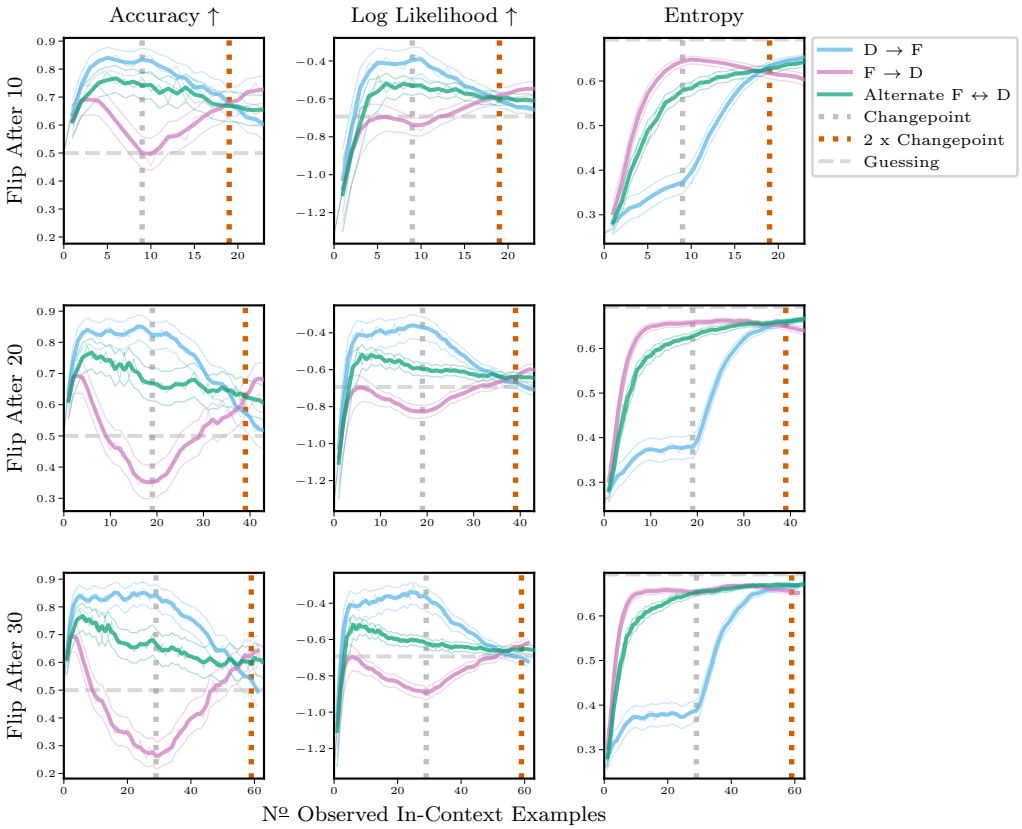


Figure D.12: Few-shot ICL on **Hate Speech** with **LLaMa-2-70B** when the **label relationship changes throughout ICL**. We here investigate **three different changepoints**, flipping labels after 10, 20, or 30 in-context observations. For (D → F), we start with default labels and change to flipped labels at the changepoint, for (F → D) we change from flipped to the default labels at the changepoint, and for (Alternating F ↔ D) we alternate between the two label relationships after every observation. For all setups, at "2 x Changepoint", the LLMs have observed the same number of examples for both label relations. If, according to NH3, ICL treats all in-context information equally, predictions should be equal at that point, regardless of whether the changepoint is after 10, 20, or 30 observations. However, this is not the case. In particular, predictions are significantly different when the changepoint is 30. Averages over 500 repetitions, we apply moving averages (window size 3), thin lines are bootstrapped 99 % confidence intervals, thin dashed lines are mean results without moving average smoothing.

References

Agarap, Abien Fred (2018). "Deep learning using rectified linear units (relu)". In: *arXiv:1803.08375*.

Agrawal, Sweta, Chunting Zhou, Mike Lewis, Luke Zettlemoyer, and Marjan Ghazvininejad (2023). "In-context examples selection for machine translation". In: *Association for Computational Linguistics (ACL)*.

Akyürek, Ekin, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou (2023). "What learning algorithm is in-context learning? Investigations with linear models". In: *International Conference on Learning Representations (ICLR)*.

Altman, Naomi S (1992). "An introduction to kernel and nearest-neighbor nonparametric regression". In: *The American Statistician*.

Amari, Shunichi (1967). "A theory of adaptive pattern classifiers". In: *IEEE Transactions on Electronic Computers*.

Anil, Rohan, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. (2023). "Palm 2 technical report". In: *arXiv:2305.10403*.

Arik, Sercan O and Tomas Pfister (2021). "Tabnet: Attentive interpretable tabular learning". In: *AAAI Conference on Artificial Intelligence (AAAI CAI)*.

Ashury-Tahan, Shir, Benjamin Sznajder, Leshem Choshen, Liat Ein-Dor, Eyal Shnarch, and Ariel Gera (2024). "Label-Efficient Model Selection for Text Generation". In: *arXiv:2402.07891*.

Atlas, Les, David Cohn, and Richard Ladner (1990). "Training Connectionist Networks with Queries and Selective Sampling". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Aytekin, Caglar (2023). "LEURN: Learning Explainable Univariate Rules with Neural Networks". In: *arXiv:2303.14937*.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer normalization". In: *arXiv:1607.06450*.

Bach, Francis (2007). "Active learning for misspecified generalized linear models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural machine translation by jointly learning to align and translate". In: *International Conference on Learning Representations (ICLR)*.

Bai, Yuntao, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. (2022a). "Training a helpful and harmless assistant with reinforcement learning from human feedback". In: *arXiv:2204.05862*.

Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. (2022b). "Constitutional ai: Harmlessness from ai feedback". In: *arXiv:2212.08073*.

Balazevic, Ivana, David Steiner, Nikhil Parthasarathy, Relja Arandjelović, and Olivier Henaff (2024). "Towards in-context scene understanding". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Barber, David (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.

Bartlett, Peter L, Philip M Long, Gábor Lugosi, and Alexander Tsigler (2020). "Benign overfitting in linear regression". In: *National Academy of Sciences*.

Bates, Stephen, Trevor Hastie, and Robert Tibshirani (2023). "Cross-validation: what does it estimate and how well does it do it?" In: *Journal of the American Statistical Association*.

Baydin, Atilim Gunes, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2018). "Automatic differentiation in machine learning: a survey". In: *Journal of Machine Learning Research*.

Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences*.

Beltagy, Iz, Matthew E. Peters, and Arman Cohan (2020). "Longformer: The Long-Document Transformer". In: *arXiv:2004.05150*.

Bengio, Y., S. Bengio, and J. Cloutier (1991). "Learning a synaptic learning rule". In: *International Joint Conference on Neural Networks (IJCNN)*.

Bengio, Yoshua, Pascal Lamblin, Dan Popovici, and Hugo Larochelle (2006). "Greedy layer-wise training of deep networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Bennett, Paul N and Vitor R Carvalho (2010). "Online stratified sampling: evaluating classifiers at web-scale". In: *International Conference on Information and Knowledge Management (CIKM)*.

Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM*.

Berger, James O and Robert L Wolpert (1988). "The likelihood principle". In: IMS.

Bernardo, José M and Adrian FM Smith (2009). *Bayesian theory*.

Beygelzimer, Alina, Sanjoy Dasgupta, and John Langford (2009). "Importance Weighted Active Learning". In: *International Conference on Machine Learning (ICML)*.

Bickford Smith, Freddie, Adam Foster, and Tom Rainforth (2024). "Making better use of unlabelled data in Bayesian active learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Bickford Smith, Freddie, Andreas Kirsch, Sebastian Farquhar, Yarin Gal, Adam Foster, and Tom Rainforth (2023). "Prediction-Oriented Bayesian Active Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Biggs, John B (1985). "The role of metalearning in study processes". In: *British journal of educational psychology*.

Billingsley, Patrick (2013). *Convergence of probability measures*. John Wiley & Sons.

Bishop, Christopher M (2006). *Pattern Recognition and Machine Learning*. Springer.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians". In: *American statistical Association*.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015a). "Weight Uncertainty in Neural Network". In: *International Conference on Machine Learning (ICML)*.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015b). "Weight uncertainty in neural network". In: *International Conference on Machine Learning (ICML)*.

Bohdal, Ondrej, Da Li, Shell Xu Hu, and Timothy Hospedales (2024). "Feed-forward latent domain adaptation". In: *Winter Conference on Applications of Computer Vision (WACV)*.

Bommasani, Rishi, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. (2021). "On the opportunities and risks of foundation models". In: *arXiv:2108.07258*.

Borgeaud, Sebastian, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. (2022). "Improving language models by retrieving from trillions of tokens". In: *International Conference on Machine Learning (ICML)*.

Borisov, Vadim, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci (2022). "Deep neural networks and tabular data: A survey". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Boser, Bernhard E, Isabelle M Guyon, and Vladimir N Vapnik (1992). "A training algorithm for optimal margin classifiers". In: *Conference on computational learning theory (COLT)*.

Boubdir, Meriem, Edward Kim, Beyza Ermis, Marzieh Fadaee, and Sara Hooker (2023). "Which Prompts Make The Difference? Data Prioritization For Efficient Human LLM Evaluation". In: *arXiv:2310.14424*.

Breiman, Leo (2001). "Random forests". In: *Machine learning*.

Breiman, Leo, Jerome Friedman, Charles J Stone, and Richard A Olshen (1984). *Classification and regression trees*.

Briol, François-Xavier, Chris J Oates, Mark Girolami, Michael A Osborne, and Dino Sejdinovic (2019). "Probabilistic integration: A role in statistical computation?" In: *Statistical Science*.

Brochu, Eric, Vlad M Cora, and Nando De Freitas (2010). "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv:1012.2599*.

Brooks, Steve, Andrew Gelman, Galin Jones, and Xiao-Li Meng (2011). *Handbook of markov chain monte carlo*. CRC press.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). "Language models are few-shot learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Bugallo, Monica F, Victor Elvira, Luca Martino, David Luengo, Joaquin Miguez, and Petar M Djuric (2017). "Adaptive importance sampling: The past, the present, and the future". In: *IEEE Signal Processing Magazine*.

Bui, Thang, Daniel Hernández-Lobato, Jose Hernandez-Lobato, Yingzhen Li, and Richard Turner (2016). "Deep Gaussian processes for regression using approximate expectation propagation". In: *International Conference on Machine Learning (ICML)*.

Buuren, Stef van and Karin Groothuis-Oudshoorn (2011). "mice: Multivariate Imputation by Chained Equations in R". In: *Journal of Statistical Software*.

Celikyilmaz, Asli, Elizabeth Clark, and Jianfeng Gao (2020). "Evaluation of text generation: A survey". In: *arXiv:2006.14799*.

Chai, Henry, Jean-Francois Ton, Michael A. Osborne, and Roman Garnett (2019). "Automated Model Selection with Bayesian Quadrature". In: *International Conference on Machine Learning (ICML)*.

Chaloner, Kathryn and Isabella Verdinelli (1995). "Bayesian experimental design: A review". In: *Statistical Science*.

Chan, Stephanie, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill (2022a). "Data distributional properties drive emergent in-context learning in transformers". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Chan, Stephanie CY, Ishita Dasgupta, Junkyung Kim, Dharshan Kumaran, Andrew K Lampinen, and Felix Hill (2022b). "Transformers generalize differently from information stored in context vs in weights". In: *arXiv:2210.05675*.

Chang, Ting-Yun and Robin Jia (2022). "Careful Data Curation Stabilizes In-context Learning". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Chapelle, Olivier, Bernhard Scholkopf, and Alexander Zien (2009). "Semi-supervised learning". In: *IEEE Transactions on Neural Networks*.

Chaudhuri, Kamalika, Sham M Kakade, Praneeth Netrapalli, and Sujay Sanghavi (2015). "Convergence rates of active learning for maximum likelihood estimation". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Chen, Mia Xu, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes (2018). "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation". In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Chen, Tianqi and Carlos Guestrin (2016). "Xgboost: A scalable tree boosting system". In: *Conference on Knowledge Discovery and Data Mining (KDD)*.

Chen, Yanda, Chen Zhao, Zhou Yu, Kathleen McKeown, and He He (2023). "On the relation between sensitivity and accuracy in in-context learning". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Chen, Yen-Chi and Youngjun Choe (2019). "Importance sampling and its optimality for stochastic simulation models". In: *Electronic Journal of Statistics*.

Chen, Yutian, Max Welling, and Alex Smola (2012). "Super-samples from kernel herding". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Child, Rewon, Scott Gray, Alec Radford, and Ilya Sutskever (2019). "Generating long sequences with sparse transformers". In: *arXiv:1904.10509*.

Choromanski, Krzysztof Marcin, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller (2021). "Rethinking Attention with Performers". In: *International Conference on Learning Representations (ICLR)*.

Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. (2022). "Palm: Scaling language modeling with pathways". In: *arXiv:2204.02311*.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Chui, Michael, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra (2018). *Notes from the AI frontier: Insights from hundreds of use cases*.

Cobb, Adam D and Brian Jalaian (2021). "Scaling Hamiltonian Monte Carlo inference for Bayesian neural networks with symmetric splitting". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Cohen, Taco S, Mario Geiger, Jonas Köhler, and Max Welling (2018). "Spherical CNNs". In: *International Conference on Learning Representations (ICLR)*.

Corneanu, Ciprian A, Sergio Escalera, and Aleix M Martinez (2020). "Computing the testing error without a testing set". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Cozad, Alison, Nikolaos V Sahinidis, and David C Miller (2014). "Learning surrogate models for simulation-based optimization". In: *AIChE Journal*.

Cramér, Harald (1999). *Mathematical methods of statistics*. Princeton university press.

Dagan, Ido, Oren Glickman, and Bernardo Magnini (2005). "The pascal recognising textual entailment challenge". In: *International Conference on Machine Learning Challenges*.

Dai, Zhenwen, Andreas Damianou, Javier González, and Neil Lawrence (2016). "Variational auto-encoded deep Gaussian processes". In: *International Conference on Learning Representations (ICLR)*.

Damianou, Andreas and Neil D Lawrence (2013). "Deep gaussian processes". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Dasgupta, Ishita, Erin Grant, and Tom Griffiths (2022). "Distinguishing rule and exemplar-based generalization in learning systems". In: *International Conference on Machine Learning (ICML)*.

Dasgupta, Sanjoy and Daniel Hsu (2008). "Hierarchical sampling for active learning". In: *International Conference on Machine Learning (ICML)*.

Dauphin, Yann N, Angela Fan, Michael Auli, and David Grangier (2017). "Language modeling with gated convolutional networks". In: *International Conference on Machine Learning (ICML)*.

Daxberger, Erik, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig (2021). "Laplace redux-effortless bayesian deep learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

De Finetti, Bruno (1929). "Funzione caratteristica di un fenomeno aleatorio". In: *Atti del Congresso Internazionale dei Matematici*.

Deng, Jia, Alex Berg, Sanjeev Satheesh, H Su, Aditya Khosla, and Li Fei-Fei (2012). *Imagenet large scale visual recognition competition (ILSVRC2012)*.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Deng, Weijian, Stephen Gould, and Liang Zheng (2021). "What does rotation prediction tell us about classifier accuracy under varying testing environments?" In: *International Conference on Machine Learning (ICML)*.

Deng, Weijian and Liang Zheng (2021). "Are labels always necessary for classifier accuracy evaluation?" In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Der Kiureghian, Armen and Ove Ditlevsen (2009). "Aleatory or epistemic? Does it matter?" In: *Structural safety*.

Devlin, Alexandra (2023). "The new steel map: reconfiguring supply chains around renewable resources". PhD thesis. University of Oxford.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

DeVries, Terrance and Graham W Taylor (2017). "Improved regularization of convolutional neural networks with cutout". In: *arXiv:1708.04552*.

Dolan, Bill and Chris Brockett (2005). "Automatically constructing a corpus of sentential paraphrases". In: *International Workshop on Paraphrasing (IWP2005)*.

Domingos, Pedro (1999). "The role of Occam's razor in knowledge discovery". In: *Data mining and knowledge discovery*.

Donahue, Jeff, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell (2014). "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International Conference on Machine Learning (ICML)*.

Doob, Joseph L (1949). "Application of the theory of martingales". In: *Le calcul des probabilites et ses applications*.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research*.

Eggensperger, Katharina, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2015). "Efficient benchmarking of hyperparameter optimizers via surrogates". In: *AAAI Conference on Artificial Intelligence (AAAI CAI)*.

Eggensperger, Katharina, Marius Lindauer, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown (2018). "Efficient benchmarking of algorithm configurators via model-based surrogates". In: *Machine Learning*.

Erhan, Dumitru, Aaron Courville, Yoshua Bengio, and Pascal Vincent (2010). "Why does unsupervised pre-training help deep learning?" In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Evans, Talfan, Shreya Pathak, Hamza Merzic, Jonathan Schwarz, Ryutaro Tanno, and Olivier J Henaff (2023). "Bad Students Make Great Teachers: Active Learning Accelerates Large-Scale Visual Understanding". In: *arXiv:2312.05328*.

Farquhar, Sebastian, Yarin Gal, and Tom Rainforth (2021). "On Statistical Bias In Active Learning: How and When to Fix It". In: *International Conference on Learning Representations (ICLR)*.

Farquhar*, Sebastian, Jannik Kossen*, Lorenz Kuhn, and Yarin Gal (2024). "Detecting hallucinations in large language models using semantic entropy". In: *Nature*.

Farquhar, Sebastian, Michael A. Osborne, and Yarin Gal (2020). "Radial Bayesian Neural Networks: Beyond Discrete Support In Large-Scale Bayesian Deep Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Ferguson, Thomas S (1973). "A Bayesian analysis of some nonparametric problems". In: *The annals of statistics*.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML)*.

Fisher, Ronald A (1912). "On an absolute criterion for fitting frequency curves". In: *Messenger of mathematics*.

Fix, Evelyn (1985). *Discriminatory analysis: nonparametric discrimination, consistency properties*.

Fong, Edwin and Chris C Holmes (2020). "On the marginal likelihood and cross-validation". In: *Biometrika*.

Fortuin, Vincent, Adrià Garriga-Alonso, Sebastian W Ober, Florian Wenzel, Gunnar Rätsch, Richard E Turner, Mark van der Wilk, and Laurence Aitchison (2022). "Bayesian neural network priors revisited". In: *International Conference on Learning Representations (ICLR)*.

Friedman, Jerome H (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*.

Fuchs, Fabian, Daniel Worrall, Volker Fischer, and Max Welling (2020). "SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Fukushima, Kunihiko (1975). "Cognitron: A self-organizing multilayered neural network". In: *Biological cybernetics*.

Gal, Yarin and Zoubin Ghahramani (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *International Conference on Machine Learning (ICML)*.

Gal, Yarin, Riashat Islam, and Zoubin Ghahramani (2017). "Deep bayesian active learning with image data". In: *International Conference on Machine Learning (ICML)*.

Ganguli, Deep, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. (2022). "Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned". In: *arXiv:2209.07858*.

Ganti, Ravi and Alexander Gray (2012). "Upal: Unbiased pool based active learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Gao, Tianyu, Adam Fisch, and Danqi Chen (2021). "Making pre-trained language models better few-shot learners". In: *Association for Computational Linguistics (ACL)*.

Garcia, Victor and Joan Bruna (2018). "Few-shot learning with graph neural networks". In: *International Conference on Learning Representations (ICLR)*.

Gardner, Jacob, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson (2018). "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Garg, Saurabh, Yifan Wu, Sivaraman Balakrishnan, and Zachary Lipton (2020). "A Unified View of Label Shift Estimation". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Garnelo, Marta, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami (2018a). "Conditional neural processes". In: *International Conference on Machine Learning (ICML)*.

Garnelo, Marta, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh (2018b). "Neural processes". In: *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*.

Gelman, Andrew (2011). *David MacKay and Occam's Razor*. URL: https://statmodeling.stat.columbia.edu/2011/12/04/david-mackay-and-occams-razor/.

Gershman, Samuel J and David M Blei (2012). "A tutorial on Bayesian nonparametric models". In: *Journal of Mathematical Psychology*.

Gibert, Ona de, Naiara Perez, Aitor García-Pablos, and Montse Cuadros (2018). "Hate Speech Dataset from a White Supremacy Forum". In: *ACL Workshop on Abusive Language Online (ALW2)*.

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Gonen, Hila, Srini Iyer, Terra Blevins, Noah A Smith, and Luke Zettlemoyer (2023). "Demystifying prompts in language models via perplexity estimation". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Gordon, Jonathan, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E Turner (2019). "Meta-learning probabilistic inference for prediction". In: *International Conference on Learning Representations (ICLR)*.

Gorishniy, Yury, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko (2024). "TabR: Tabular Deep Learning Meets Nearest Neighbors". In: *International Conference on Learning Representations (ICLR)*.

Gotovos, Alkis (2013). "Active learning for level set estimation". In: *International Joint Conference on Artificial Intelligence (IJCAI)*.

Goyal, Priya, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He (2017). "Accurate, large minibatch sgd: Training imagenet in 1 hour". In: *arXiv:1706.02677*.

Graves, Alex (2011). "Practical variational inference for neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Griewank, Andreas (2012). "Who invented the reverse mode of differentiation". In: *Documenta Mathematica, Extra Volume ISMP*.

Grinsztajn, Léo, Edouard Oyallon, and Gaël Varoquaux (2022). "Why do tree-based models still outperform deep learning on typical tabular data?" In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Guerra, Silvio B, Ricardo BC Prudêncio, and Teresa B Ludermir (2008). "Predicting the performance of learning algorithms using support vector machines as meta-regressors". In: *International Conference on Artificial Neural Networks (ICANN)*.

Guu, Kelvin, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang (2018). "Generating sentences by editing prototypes". In: *Transactions of the Association for Computational Linguistics*.

Guu, Kelvin, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang (2020). "Realm: Retrieval-augmented language model pre-training". In: *International Conference on Machine Learning (ICML)*.

Hahn, Michael and Navin Goyal (2023). "A theory of emergent in-context learning as implicit structure induction". In: *arXiv:2303.07971*.

Han, Chi, Ziqi Wang, Han Zhao, and Heng Ji (2023). "In-Context Learning of Large Language Models Explained as Kernel Regression". In: *arXiv:2305.12766*.

Hanneke, Steve (2011a). "Adaptive Rates of Convergence in Active Learning". In: *Annals of Statistics*.

Hanneke, Steve (2011b). "Rates of convergence in active learning". In: *The Annals of Statistics*.

Hanson, Stephen and Lorien Pratt (1988). "Comparing biases for minimal network construction with back-propagation". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Hashimoto, Tatsunori B, Kelvin Guu, Yonatan Oren, and Percy Liang (2018). "A retrieve-and-edit framework for predicting structured outputs". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Hastie, Trevor, Andrea Montanari, Saharon Rosset, and Ryan J Tibshirani (2022). "Surprises in high-dimensional ridgeless least squares interpolation". In: *Annals of statistics*.

He, Kaiming, Ross Girshick, and Piotr Dollár (2019). "Rethinking imagenet pre-training". In: *International Conference on Computer Vision (ICCV)*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, Xin, Kaiyong Zhao, and Xiaowen Chu (2021). "AutoML: A survey of the state-of-the-art". In: *Knowledge-Based Systems*.

Hendel, Roee, Mor Geva, and Amir Globerson (2023). "In-context learning creates task vectors". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Hendrycks, Dan and Kevin Gimpel (2016). "Gaussian error linear units (gelus)". In: *arXiv:1606.08415*.

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (2012). *Neural networks for machine learning. Lecture 6a: overview of mini-batch gradient descent*.

Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh (2006). "A fast learning algorithm for deep belief nets". In: *Neural computation*.

Hinton, Geoffrey E and Drew Van Camp (1993). "Keeping the neural networks simple by minimizing the description length of the weights". In: *Conference on computational learning theory (COLT)*.

Ho, Jonathan, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans (2019). "Axial attention in multidimensional transformers". In: *arXiv:1912.12180*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation*.

Hoffmann, Jordan, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. (2022). "Training compute-optimal large language models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Hollmann, Noah, Samuel Müller, Katharina Eggensperger, and Frank Hutter (2023). "Tabpfn: A transformer that solves small tabular classification problems in a second". In: *International Conference on Learning Representations (ICLR)*.

Honaker, James and Gary King (2010). "What to do About Missing Values in Time Series Cross-Section Data". In: *American Journal of Political Science*.

Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly (2019). "Parameter-efficient transfer learning for NLP". In: *International Conference on Machine Learning (ICML)*.

Houlsby, Neil, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel (2011). "Bayesian active learning for classification and preference learning". In: *arXiv:1112.5745*.

Howard, Andrew G, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv:1704.04861*.

Howard, Jeremy and Sebastian Ruder (2018). "Universal language model fine-tuning for text classification". In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Hu, Edward J, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2021). "Lora: Low-rank adaptation of large language models". In: *International Conference on Learning Representations (ICLR)*.

Hu, Qiang, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Lei Ma, Mike Papadakis, and Yves Le Traon (2018). "Test Optimization in DNN Testing: A Survey". In: *ACM Transactions on Software Engineering and Methodology*.

Huszár, Ferenc (2023). *Implicit Bayesian Inference in Large Language Models*.

Hutchinson, Ben, Negar Rostamzadeh, Christina Greer, Katherine Heller, and Vinodkumar Prabhakaran (2022). "Evaluation gaps in machine learning practice". In: *ACM Conference on Fairness, Accountability, and Transparency*.

Hutchinson, Michael, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim (2021). "LieTransformer: Equivariant self-attention for Lie Groups". In: *International Conference on Machine Learning (ICML)*.

Imberg, Henrik, Johan Jonasson, and Marina Axelson-Fisk (2020). "Optimal Sampling in Unbiased Active Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Inc., Google (2021). "Kaggle". In: *https://www.kaggle.com/*.

Ivakhnenko, Alexey Grigorevich (1971). "Polynomial theory of complex systems". In: *IEEE transactions on Systems, Man, and Cybernetics*.

Izmailov, Pavel, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Wilson (2021). "What are Bayesian neural network posteriors really like?" In: *International Conference on Machine Learning (ICML)*.

Jaegle, Andrew, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. (2022). "Perceiver io: A general architecture for structured inputs & outputs". In: *International Conference on Learning Representations (ICLR)*.

Jaegle, Andrew, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira (2021). "Perceiver: General Perception with Iterative Attention". In: *International Conference on Machine Learning (ICML)*.

Jain, Neel, Khalid Saifullah, Yuxin Wen, John Kirchenbauer, Manli Shu, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein (2023). "Bring your own data! self-supervised evaluation for large language models". In: *arXiv:2306.13651*.

Jaynes, Edwin T (2003). *Probability theory: The logic of science*. Cambridge University Press.

Jesson, Andrew, Panagiotis Tigas, Joost Van Amersfoort, Andreas Kirsch, Uri Shalit, and Yarin Gal (2021). "Causal-bald: Deep bayesian active learning of outcomes to infer treatment-effects from observational data". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Ji, Disi, Robert L. Logan, Padhraic Smyth, and Mark Steyvers (2021). "Active Bayesian Assessment of Black-Box Classifiers". In: *AAAI Conference on Artificial Intelligence (AAAI CAI)*.

Jiang, Hui (2023). "A latent space theory for emergent abilities in large language models". In: *arXiv:2304.09960*.

Jin, Qixuan and Talip Ucar (2023). "Benchmarking tabular representation models in transfer learning settings". In: *NeurIPS 2023 Second Table Representation Learning Workshop*.

Jordan, Michael I, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul (1999). "An introduction to variational methods for graphical models". In: *Machine learning*.

Joulin, Armand, Laurens Van Der Maaten, Allan Jabri, and Nicolas Vasilache (2016). "Learning visual features from large weakly supervised data". In: *European Conference on Computer Vision (ECCV)*.

Kadavath, Saurav, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. (2022). "Language models (mostly) know what they know". In: *arXiv:2207.05221*.

Kahn, Herman (1955). *Use of different Monte Carlo sampling techniques*. Rand Corporation.

Kahn, Herman and Andy W Marshall (1953). "Methods of reducing sample size in Monte Carlo computations". In: *Journal of the Operations Research Society of America*.

Kanagawa, Motonobu and Philipp Hennig (2019). "Convergence guarantees for adaptive Bayesian quadrature methods". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Karamchandani, A, P Bjerager, and CA Cornell (1989). "Adaptive importance sampling". In: *Structural Safety and Reliability*. ASCE.

Katariya, Namit, Arun Iyer, and Sunita Sarawagi (2012). "Active evaluation of classifiers on large datasets". In: *International Conference on Data Mining (ICDM)*.

Katharopoulos, Angelos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret (2020). "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention". In: *International Conference on Machine Learning (ICML)*.

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu (2017). "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Ke, Nan Rosemary, Silvia Chiappa, Jane Wang, Anirudh Goyal, Jorg Bornschein, Melanie Rey, Theophane Weber, Matthew Botvinic, Michael Mozer, and Danilo Jimenez Rezende (2023). "Learning to induce causal structure". In: *International Conference on Learning Representations (ICLR)*.

Kendall, Alex and Yarin Gal (2017). "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kim, Hyunjik, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh (2019). "Attentive Neural Processes". In: *International Conference on Learning Representations (ICLR)*.

Kim, Yoon-Yeong, Youngjae Cho, JoonHo Jang, Byeonghu Na, Yeongmin Kim, Kyungwoo Song, Wanmo Kang, and Il-Chul Moon (2023). "SAAL: sharpness-aware active learning". In: *International Conference on Machine Learning (ICML)*.

Kimeldorf, George and Grace Wahba (1971). "Some results on Tchebycheffian spline functions". In: *Journal of mathematical analysis and applications*.

King, Gary, James Honaker, Anne Joseph, and Kenneth Scheve (2001). "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation". In: *American Political Science Review*.

Kingma, Diederik P and Jimmy Ba (2015). "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations (ICLR)*.

Kingma, Diederik P, Danilo J Rezende, Shakir Mohamed, and Max Welling (2014). "Semi-supervised learning with deep generative models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kipf, Thomas, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel (2018). "Neural relational inference for interacting systems". In: *International Conference on Machine Learning (ICML)*.

Kipf, Thomas and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*.

Kirsch, Andreas, Joost Van Amersfoort, and Yarin Gal (2019). "Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kirsch, Louis, James Harrison, Jascha Sohl-Dickstein, and Luke Metz (2022). "General-purpose in-context learning by meta-learning transformers". In: *arXiv:2212.04458*.

Kleijnen, Jack PC (2009). "Kriging metamodeling in simulation: A review". In: *European journal of operational research (EJOR)*.

Kolesnikov, Alexander, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby (2020). "Big transfer (bit): General visual representation learning". In: *Proceedings of the European conference on computer vision (ECCV)*.

Kossen*, Jannik, Neil Band*, Clare Lyle, Aidan N Gomez, Tom Rainforth, and Yarin Gal (2021). "Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kossen, Jannik, Cătălina Cangea, Eszter Vértes, Andrew Jaegle, Viorica Patraucean, Ira Ktena, Nenad Tomasev, and Danielle Belgrave (2023). "Active Acquisition for Multimodal Temporal Data: A Challenging Decision-Making Task". In: *Transactions on Machine Learning Research (TMLR)*.

Kossen*, Jannik, Mark Collier*, Basil Mustafa, Xiao Wang, Xiaohua Zhai, Lucas Beyer, Andreas Steiner, Jesse Berent, Rodolphe Jenatton, and Efi Kokiopoulou (2023). "Three Towers: Flexible Contrastive Learning with Pretrained Image Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kossen*, Jannik, Sebastian Farquhar*, Yarin Gal, and Tom Rainforth (2021). "Active Testing: Sample-Efficient Model Evaluation". In: *International Conference on Machine Learning (ICML)*.

Kossen, Jannik, Sebastian Farquhar, Yarin Gal, and Tom Rainforth (2022). "Active Surrogate Estimators: An Active Learning Approach to Label-Efficient Model Evaluation". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kossen, Jannik, Yarin Gal, and Tom Rainforth (2024). "In-Context Learning Learns Label Relationships but Is Not Conventional Learning". In: *International Conference on Learning Representations (ICLR)*.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). *Learning multiple layers of features from tiny images*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Kumar, Anurag and Bhiksha Raj (2018). "Classifier risk estimation under limited labeling resources". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*.

Lake, Brenden M., Ruslan Salakhutdinov, and Joshua B. Tenenbaum (2015). "Human-level concept learning through probabilistic program induction". In: *Science*.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Lan, Zhenzhong, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut (2020). "Albert: A lite bert for self-supervised learning of language representations". In: *International Conference on Learning Representations (ICLR)*.

Lawrence, Neil, Matthias Seeger, and Ralf Herbrich (2002). "Fast sparse Gaussian process methods: The informative vector machine". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature*.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*.

Lee, Juho, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh (2019). "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International Conference on Machine Learning (ICML)*.

Leshno, Moshe, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken (1993). "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural networks*.

Levesque, Hector, Ernest Davis, and Leora Morgenstern (2012). "The winograd schema challenge". In: *International Conference on the principles of knowledge representation and reasoning*.

Lewis, David D (1995). "A sequential algorithm for training text classifiers: Corrigendum and additional data". In: *Acm Sigir Forum*, pp. 13–19.

Li, Weizhi, Gautam Dasarathy, Karthikeyan Natesan Ramamurthy, and Visar Berisha (2022). "A label efficient two-sample test". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Li, Xiang Lisa and Percy Liang (2021). "Prefix-tuning: Optimizing continuous prompts for generation". In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Li, Xiaonan and Xipeng Qiu (2023). "Finding supporting examples for in-context learning". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Li, Zhuohan, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez (2020). "Train big, then compress: Rethinking model size for efficient training and inference of transformers". In: *International Conference on Machine Learning (ICML)*.

Liang, Percy, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. (2023). "Holistic evaluation of language models". In: *Transactions on Machine Learning Research (TMLR)*.

Liao, Jay Chiehen and Cheng-Te Li (2023). "TabGSL: Graph Structure Learning for Tabular Data Prediction". In: *arXiv:2305.15843*.

Lin, Stephanie, Jacob Hilton, and Owain Evans (2023). "Teaching models to express their uncertainty in words". In: *TMLR*.

Lindley, Dennis V (1956). "On a measure of the information provided by an experiment". In: *The Annals of Mathematical Statistics*.

Linnainmaa, Seppo (1970). "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". PhD thesis. Master's Thesis (in Finnish), Univ. Helsinki.

Linnainmaa, Seppo (1976). "Taylor expansion of the accumulated rounding error". In: *BIT Numerical Mathematics*.

Liu, Jeremiah, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan (2020). "Simple and principled uncertainty estimation with deterministic deep learning via distance awareness". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Liu, Jiachang, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen (2022). "What Makes Good In-Context Examples for GPT-3?" In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Liu, Pengfei, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig (2023a). "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing". In: *ACM Computing Surveys*.

Liu, Peter J, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer (2018). "Generating wikipedia by summarizing long sequences". In: *International Conference on Learning Representations (ICLR)*.

Liu, Sheng, Lei Xing, and James Zou (2023b). "In-context vectors: Making in context learning more effective and controllable through latent space steering". In: *arXiv:2311.06668*.

Liu, T., A. Moore, and A. Gray (2006). "New Algorithms for Efficient High-Dimensional Nonparametric Classification". In: *Journal of Machine Learning Research (JMLR)*.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov (2019). "Roberta: A robustly optimized bert pretraining approach". In: *arXiv:1907.11692*.

Locatello, Francesco, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf (2020). "Object-Centric Learning with Slot Attention". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Loshchilov, Ilya and Frank Hutter (2017). "Sgdr: Stochastic gradient descent with warm restarts". In: *International Conference on Learning Representations (ICLR)*.

Loshchilov, Ilya and Frank Hutter (2019). "Decoupled weight decay regularization". In: *International Conference on Learning Representations (ICLR)*.

Lotfi, Sanae, Pavel Izmailov, Gregory Benton, Micah Goldblum, and Andrew Gordon Wilson (2022). "Bayesian model selection, the marginal likelihood, and generalization". In: *International Conference on Machine Learning (ICML)*.

Lowell, David, Zachary C. Lipton, and Byron C. Wallace (2019). "Practical Obstacles to Deploying Active Learning". In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Lu, Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis (2019). "Dying relu and initialization: Theory and numerical examples". In: *arXiv:1903.06733*.

Lu, Yao, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp (2022). "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity". In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Luo, Jiaqi and Shixin Xu (2023). "NCART: Neural Classification and Regression Tree for Tabular Data". In: *arXiv:2307.12198*.

MacKay, David JC (1992a). "A practical Bayesian framework for backpropagation networks". In: *Neural computation*.

MacKay, David JC (1992b). "Bayesian interpolation". In: *Neural computation*.

MacKay, David JC (1992c). "Information-based objective functions for active data selection". In: *Neural computation*.

MacKay, David JC (1992d). "The evidence framework applied to classification networks". In: *Neural computation*.

MacKay, David JC (2003). *Information theory, inference and learning algorithms*. Cambridge University Press.

Maddox, Wesley J, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson (2019). "A simple baseline for bayesian uncertainty in deep learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Mahajan, Dhruv, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten (2018). "Exploring the limits of weakly supervised pretraining". In: *Proceedings of the European conference on computer vision (ECCV)*.

Malo, P., A. Sinha, P. Korhonen, J. Wallenius, and P. Takala (2014). "Good debt or bad debt: Detecting semantic orientations in economic texts". In: *Journal of the Association for Information Science and Technology*.

Malte, Aditya and Pratik Ratadiya (2019). "Evolution of transfer learning in natural language processing". In: *arXiv:1910.07370*.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Marton, Sascha, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt (2024). "GRANDE: Gradient-Based Decision Tree Ensembles". In: *International Conference on Learning Representations (ICLR)*.

Matsuura, Mitsuru and Satoshi Hara (2023). "Active Model Selection: A Variance Minimization Approach". In: *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*.

McCreery, Clara H., Namit Katariya, Anitha Kannan, Manish Chablani, and Xavier Amatriain (2020). "Effective Transfer Learning for Identifying Similar Questions: Matching User Questions to COVID-19 FAQs". In: *Conference on Knowledge Discovery and Data Mining (KDD)*.

Metropolis, Nicholas, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics*.

Min, Sewon, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi (2022a). "Metaicl: Learning to learn in context". In: *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Min, Sewon, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer (2022b). "Rethinking the role of demonstrations: What makes in-context learning work?" In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Mindermann, Sören, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, et al. (2022). "Prioritized training on points that are learnable, worth learning, and not yet learnt". In: *International Conference on Machine Learning (ICML)*.

Mitchell, Tom M (1980). *The need for biases in learning generalizations*.

Morgan, Nelson and Hervé Bourlard (1989). "Generalization and parameter estimation in feedforward nets: Some experiments". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Moulines, Eric and Francis Bach (2011). "Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger.

Movshovitz-Attias, Yair, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh (2017). "No fuss distance metric learning using proxies". In: *International Conference on Computer Vision (ICCV)*.

Müller, Samuel, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter (2022). "Transformers can do bayesian inference". In: *International Conference on Learning Representations (ICLR)*.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. MIT press.

Murphy, Kevin P (2022). *Probabilistic machine learning: an introduction*. MIT press.

Murphy, Kevin P (2023). *Probabilistic machine learning: Advanced Topics*. MIT press.

Muslea, Ion, Steven Minton, and Craig A Knoblock (2002). "Active+ semi-supervised learning= robust multi-view learning". In: *ICML*.

Nadaraya, Elizbar A (1964). "On estimating regression". In: *Theory of Probability & Its Applications*.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *International Conference on Machine Learning (ICML)*.

Narang, Sharan, Hyung Won Chung, Yi Tay, William Fedus, Thibault Févry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel (2021). "Do Transformer Modifications Transfer Across Implementations and Applications?" In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Nayak, Nihal V, Ethan R Elenberg, and Clemens Rosenbaum (2022). "CEREAL: Few-Sample Clustering Evaluation". In: *arXiv:2210.00064*.

Neal, Radford M (1995). *Bayesian learning for neural networks*.

Neyshabur, Behnam, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro (2019). "The role of over-parametrization in generalization of neural networks". In: *International Conference on Learning Representations (ICLR)*.

Nguyen, Phuc, Deva Ramanan, and Charless Fowlkes (2018). "Active Testing: An Efficient and Robust Framework for Estimating Accuracy". In: *International Conference on Machine Learning (ICML)*.

Nguyen, Tung and Aditya Grover (2022). "Transformer neural processes: Uncertainty-aware meta learning via sequence modeling". In: *International Conference on Machine Learning (ICML)*.

Notin, Pascal, Ruben Weitzman, Debora S Marks, and Yarin Gal (2023). "ProteinNPT: Improving Protein Property Prediction and Design with Non-Parametric Transformers". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

O'Hagan, Anthony (1991). "Bayes–hermite quadrature". In: *Journal of statistical planning and inference*.

O'Neill, Ben (2009). "Exchangeability, correlation, and Bayes' effect". In: *International statistical review*.

Ochiai, Takuma, Keiichiro Seno, Kota Matsui, and Satoshi Hara (2023). "Active Testing of Binary Classification Model Using Level Set Estimation". In: *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*.

Oh, Man-Suk and James O Berger (1992). "Adaptive importance sampling in Monte Carlo integration". In: *Journal of Statistical Computation and Simulation*.

Olah, Chris, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev (2018). "The building blocks of interpretability". In: *Distill*.

Osband, Ian, Seyed Mohammad Asghari, Benjamin Van Roy, Nat McAleese, John Aslanides, and Geoffrey Irving (2023). "Fine-Tuning Language Models via Epistemic Neural Networks". In: *International Conference on Machine Learning (ICML)*.

Osborne, Michael (2010). “Bayesian Gaussian processes for sequential prediction, optimisation and quadrature”. PhD thesis. Oxford University, UK.

Osborne, Michael, Roman Garnett, Zoubin Ghahramani, David K Duvenaud, Stephen J Roberts, and Carl Rasmussen (2012). “Active Learning of Model Evidence Using Bayesian Quadrature”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Oswald, Johannes von, Eyvind Niklasson, Maximilian Schlegel, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, Razvan Pascanu, et al. (2023). “Uncovering mesa-optimization algorithms in transformers”. In: *arXiv:2309.05858*.

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. (2022). “Training language models to follow instructions with human feedback”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Owen, Art and Yi Zhou (2000). “Safe and effective importance sampling”. In: *Journal of the American Statistical Association*.

Owen, Art B (2013). *Monte Carlo theory, methods and examples*.

Pan, Jane, Tianyu Gao, Howard Chen, and Danqi Chen (2023). “What In-Context Learning "Learns" In-Context: Disentangling Task Recognition and Task Learning”. In: *Association for Computational Linguistics (ACL)*.

Parmar, Niki, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran (2018). “Image transformer”. In: *International Conference on Machine Learning (ICML)*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Pawelczyk, Martin, Seth Neel, and Himabindu Lakkaraju (2023). “In-context unlearning: Language models as few shot unlearners”. In: *arXiv:2310.07579*.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research (JMLR)*.

Perez, Ethan, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving (2022). “Red teaming language models with language models”. In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Petrov, Aleksandar, Philip HS Torr, and Adel Bibi (2024). “When do prompting and prefix-tuning work? a theory of capabilities and limitations”. In: *International Conference on Learning Representations (ICLR)*.

Platform, Google Cloud AI (2021). *Getting started with the built-in TabNet algorithm*.

Polo, Felipe Maia, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin (2024). “tinyBenchmarks: evaluating LLMs with fewer examples”. In: *arXiv:2402.14992*.

Popov, Sergei, Stanislav Morozov, and Artem Babenko (2020). "Neural oblivious decision ensembles for deep learning on tabular data". In: *International Conference on Learning Representations (ICLR)*.

Pourkamali-Anaraki, Farhad, Tahamina Nasrin, Robert E Jensen, Amy M Peterson, and Christopher J Hansen (2023). "Evaluation of classification models in limited data scenarios with application to additive manufacturing". In: *Engineering Applications of Artificial Intelligence*.

Prabhu, Ameya, Vishaal Udandarao, Philip Torr, Matthias Bethge, Adel Bibi, and Samuel Albanie (2024). "Lifelong Benchmarks: Efficient Model Evaluation in an Era of Rapid Progress". In: *arXiv:2402.19472*.

Prokhorenkova, Liudmila, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin (2018). "CatBoost: unbiased boosting with categorical features". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett.

Qian, Zhiguang, Carolyn Conner Seepersad, V Roshan Joseph, Janet K Allen, and CF Jeff Wu (2005). "Building surrogate models based on detailed and approximate simulations". In: *Journal of Mechanical Design*.

Qiu, Xipeng, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang (2020). "Pre-trained models for natural language processing: A survey". In: *Science China Technological Sciences*.

Rabbani, Shourav B, Ivan V Medri, and Manar D Samad (2024). "Attention versus Contrastive Learning of Tabular Data–A Data-centric Benchmarking". In: *arXiv:2401.04266*.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. (2021). "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning (ICML)*.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). "Improving language understanding with unsupervised learning". In: *Technical report, OpenAI*.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language models are unsupervised multitask learners". In: *OpenAI blog*.

Rainforth, Thomas (2017). "Automating inference, learning, and design using probabilistic programming". PhD thesis. University of Oxford.

Rainforth, Tom, Adam Foster, Desi R Ivanova, and Freddie Bickford Smith (2024). "Modern bayesian experimental design". In: *Statistical Science*.

Raj, Anant and Francis Bach (2022). "Convergence of Uncertainty Sampling for Active Learning". In: *International Conference on Machine Learning (ICML)*.

Ramachandran, Prajit, Barret Zoph, and Quoc V Le (2017). "Searching for activation functions". In: *arXiv:1710.05941*.

Rao, Roshan M, Jason Liu, Robert Verkuil, Joshua Meier, John Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives (2021). "MSA transformer". In: *International Conference on Machine Learning (ICML)*.

Rasmussen, Carl Edward (2003). "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer.

Rasmussen, Carl Edward and Zoubin Ghahramani (2003). "Bayesian Monte Carlo". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Rastogi, Richa, Yair Schiff, Alon Hacohen, Zhaozhi Li, Ian Lee, Yuntian Deng, Mert R Sabuncu, and Volodymyr Kuleshov (2023). "Semi-parametric inducing point networks and neural processes". In: *International Conference on Learning Representations (ICLR)*.

Razeghi, Yasaman, Robert L Logan IV, Matt Gardner, and Sameer Singh (2022). "Impact of pretraining term frequencies on few-shot reasoning". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Redyuk, Sergey, Sebastian Schelter, Tammo Rukat, Volker Markl, and Felix Biessmann (2019). "Learning to validate the predictions of black box machine learning models on unseen data". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*.

Ren, Pengzhen, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang (2021). "A survey of deep active learning". In: *ACM computing surveys (CSUR)*.

Rice, Leslie, Eric Wong, and Zico Kolter (2020). "Overfitting in adversarially robust deep learning". In: *International Conference on Machine Learning (ICML)*.

Robert, Christian P (2007). *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review*.

Roweis, Sam, Geoffrey Hinton, and Ruslan Salakhutdinov (2004). "Neighbourhood component analysis". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Rubachev, Ivan, Artem Alekberov, Yury Gorishniy, and Artem Babenko (2022). "Revisiting pretraining objectives for tabular deep learning". In: *arXiv:2207.03208*.

Ruder, Sebastian, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf (2019). "Transfer learning in natural language processing". In: *North American Chapter of the Association for Computational Linguistics (NAACL): Tutorials*.

Ruder, Sebastian, Jonas Pfeiffer, and Ivan Vulić (2022). "Modular and Parameter-Efficient Fine-Tuning for NLP Models". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams, et al. (1985). *Learning internal representations by error propagation*.

Ruppert, David, Matt P Wand, and Raymond J Carroll (2003). *Semiparametric regression*. Cambridge University Press.

Sabato, Sivan and Remi Munos (2014). "Active regression by stratification". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Salimbeni, Hugh and Marc Deisenroth (2017). "Doubly Stochastic Variational Inference for Deep Gaussian Processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett.

Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv:1910.01108*.

Sawade, Christoph, Niels Landwehr, Steffen Bickel, and Tobias Scheffer (2010). "Active risk estimation". In: *International Conference on Machine Learning (ICML)*.

Schäfl, Bernhard, Lukas Gruber, Angela Bitto-Nemling, and Sepp Hochreiter (2022). "Hopular: Modern hopfield networks for tabular data". In: *arXiv:2206.00664*.

Schapire, Robert E (1990). "The strength of weak learnability". In: *Machine learning*.

Schmidhuber, Jürgen (2022). "Annotated history of modern AI and Deep learning". In: *arXiv:2212.11279*.

Schölkopf, Bernhard, Ralf Herbrich, and Alex J Smola (2001). "A generalized representer theorem". In: *Conference on computational learning theory (COLT)*.

Sebastiani, Paola and Henry P Wynn (2000). "Maximum entropy sampling and optimal Bayesian experimental design". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Seidenschwarz, Jenny, Ismail Elezi, and Laura Leal-Taixé (2021). "Learning Intra-Batch Connections for Deep Metric Learning". In: *International Conference on Machine Learning (ICML)*.

Settles, Burr (2010). "Active Learning Literature Survey". In: *Machine Learning*.

Seung, H Sebastian, Manfred Opper, and Haim Sompolinsky (1992). "Query by committee". In: *Workshop on Computational learning theory*.

Shahriari, Bobak, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas (2015). "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE*.

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Shannon, Claude Elwood (1948). "A mathematical theory of communication". In: *The Bell system technical journal*.

Shazeer, Noam (2020). "Glu variants improve transformer". In: *arXiv:2002.05202*.

Shwartz-Ziv, Ravid and Amitai Armon (2022). "Tabular data: Deep learning is not all you need". In: *Information Fusion*.

Si, Chenglei, Dan Friedman, Nitish Joshi, Shi Feng, Danqi Chen, and He He (2023). "Measuring Inductive Biases of In-Context Learning with Underspecified Demonstrations". In: *Association for Computational Linguistics (ACL)*.

Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts (2013). "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Somepalli, Gowthami, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein (2021). "Saint: Improved neural networks for tabular data via row attention and contrastive pre-training". In: *arXiv:2106.01342*.

Song, Weiping, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang (2019). "Autoint: Automatic feature interaction learning via self-attentive neural networks". In: *International Conference on Information and Knowledge Management (CIKM)*.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of machine learning research*.

Stamatatos, Efstathios (2009). "A survey of modern authorship attribution methods". In: *American Society for information Science and Technology*.

Stekhoven, D.J. and P. Buehlmann (2012). "MissForest - nonparametric missing value imputation for mixed-type data". In: *Bioinformatics*.

Stone, Mervyn (1974). "Cross-validatory choice and assessment of statistical predictions". In: *Journal of the royal statistical society: Series B (Methodological)*.

Su, Yu-Sung, Andrew E. Gelman, Jennifer Hill, and Masanao Yajima (2012). "Multiple Imputation with Diagnostics (mi) in R: Opening Windows into the Black Box". In: *Journal of Statistical Software*.

Sugiyama, Masashi (2006). "Active Learning for Misspecified Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Sun, Chen, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta (2017). "Revisiting unreasonable effectiveness of data in deep learning era". In: *Proceedings of the IEEE international conference on computer vision*.

Sun, Xiaoxiao, Yunzhong Hou, Hongdong Li, and Liang Zheng (2021). "Label-Free Model Evaluation with Semi-Structured Dataset Representations". In: *arXiv:2112.00694*.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Talagala, Thiyanga S, Feng Li, and Yanfei Kang (2022). "FFORMPP: Feature-based forecast model performance prediction". In: *International Journal of Forecasting*.

Tay, Yi, Mostafa Dehghani, Dara Bahri, and Donald Metzler (2020). "Efficient Transformers: A Survey". In: *ACM Computing Surveys*.

Taylor, Wilson L (1953). ""Cloze procedure": A new tool for measuring readability". In: *Journalism quarterly*.

Thimonier, Hugo, Fabrice Popineau, Arpad Rimmel, and Bich-Liên Doan (2023). "Beyond individual input for deep anomaly detection on tabular data". In: *arXiv:2305.15121*.

Thompson, William R (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika*.

TII, Technology Innovation Institute (2023). *Falcon LLM*.

Todd, Eric, Millicent L Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau (2024). "Function vectors in large language models". In: *International Conference on Learning Representations (ICLR)*.

Tolstikhin, Ilya O, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. (2021). "Mlp-mixer: An all-mlp architecture for vision". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. (2023a). "Llama: Open and efficient foundation language models". In: *arXiv:2302.13971*.

Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. (2023b). "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv:2307.09288*.

Tran, Ba-Hien, Simone Rossi, Dimitrios Milios, and Maurizio Filippone (2022). "All you need is a good functional prior for Bayesian deep learning". In: *Journal of Machine Learning Research (JMLR)*.

Trottier, Ludovic, Philippe Giguere, and Brahim Chaib-Draa (2017). "Parametric exponential linear unit for deep convolutional neural networks". In: *IEEE International Conference on Machine Learning and applications (ICMLA)*.

Van Amersfoort, Joost, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal (2021). "On feature collapse and deep kernel learning for single forward pass uncertainty". In: *arXiv:2102.11409*.

Van Amersfoort, Joost, Lewis Smith, Yee Whye Teh, and Yarin Gal (2020). "Uncertainty estimation using a single deep deterministic neural network". In: *International Conference on Machine Learning (ICML)*.

Van Amersfoort, Joost René (2022). "Uncertainty Estimation: single forward pass methods and applications in Active Learning". PhD thesis. University of Oxford.

Van der Vaart, Aad W (2000). *Asymptotic statistics*. Cambridge University Press.

VanderPlas, Jake (2014). "Frequentism and bayesianism: a python-driven primer". In: *arXiv:1411.5018*.

Vanschoren, Joaquin (2018). "Meta-learning: A survey". In: *arXiv:1810.03548*.

Vapnik, Vladimir (1991). "Principles of risk minimization for learning theory". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Vapnik, Vladimir (2006). *Estimation of dependences based on empirical data*.

Vapnik, Vladimir and Alexey Chervonenkis (1974). *Theory of pattern recognition*. Nauka, Moscow.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio (2018). "Graph attention networks". In: *International Conference on Learning Representations (ICLR)*.

Vijayakumar, Sethu and Stefan Schaal (1997). "Local dimensionality reduction for locally weighted learning". In: *International Symposium on Computational Intelligence in Robotics and Automation*. IEEE.

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. (2016). "Matching networks for one shot learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Vishwakarma, Harit, Heguang Lin, Frederic Sala, and Ramya Korlakai Vinayak (2023). "Promises and Pitfalls of Threshold-based Auto-labeling". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Vladymyrov, Max, Johannes von Oswald, Mark Sandler, and Rong Ge (2024). "Linear Transformers are Versatile In-Context Learners". In: *arXiv:2402.14180*.

Von Oswald, Johannes, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov (2023). "Transformers learn in-context by gradient descent". In: *International Conference on Machine Learning (ICML)*.

Wald, Abraham (1949). "Note on the consistency of the maximum likelihood estimate". In: *The Annals of Mathematical Statistics*.

Wang, Alan Q and Mert R Sabuncu (2023). "A flexible Nadaraya-Watson head can offer explainable and calibrated classification". In: *Transactions on Machine Learning Research (TMLR)*.

Wang, Alex, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman (2019a). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *International Conference on Learning Representations (ICLR)*.

Wang, Sida I and Christopher D Manning (2012). "Baselines and bigrams: Simple, good sentiment and topic classification". In: *Association for Computational Linguistics (ACL)*.

Wang, Xinshao, Yang Hua, Elyor Kodirov, Guosheng Hu, Romain Garnier, and Neil M Robertson (2019b). "Ranked list loss for deep metric learning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wei, Jerry, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. (2023). "Larger language models do in-context learning differently". In: *arXiv:2303.03846*.

Wenzel, Florian, Kevin Roth, Bastiaan S Veeling, Jakub Świątkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin (2020). "How good is the bayes posterior in deep neural networks really?" In: *International Conference on Machine Learning (ICML)*.

Wies, Noam, Yoav Levine, and Amnon Shashua (2023). "The learnability of in-context learning". In: *arXiv:2303.07895*.

Wild, Veit David, Sahra Ghalebikesabi, Dino Sejdinovic, and Jeremias Knoblauch (2024). "A rigorous link between deep ensembles and (variational) Bayesian methods". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Williams, Ronald J and David Zipser (1989). "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation*.

Wilson, Andrew G and Pavel Izmailov (2020). "Bayesian deep learning and a probabilistic perspective of generalization". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Wilson, Andrew Gordon, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing (2016). "Deep Kernel Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. (2020). "Huggingface's transformers: State-of-the-art natural language processing". In: *EMNLP: System Demonstrations*.

Wörmann, Julian, Daniel Bogdoll, Etienne Bührle, Han Chen, Evaristus Fuh Chuo, Kostadin Cvejoski, Ludger van Elst, Philip Gottschall, Stefan Griesche, Christian Hellert, et al. (2022). "Knowledge augmented machine learning with applications in autonomous driving: A survey". In: *arXiv:2205.04712*.

Wu, Zhaofeng, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim (2023). "Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks". In: *arXiv:2307.02477*.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*.

Xie, Sang Michael, Aditi Raghunathan, Percy Liang, and Tengyu Ma (2022). "An explanation of in-context learning as implicit bayesian inference". In: *International Conference on Learning Representations (ICLR)*.

Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations (ICLR)*.

Yamaguchi, Kouichi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto (1990). "A neural network for speaker-independent isolated word recognition". In: *International Conference on Spoken Language Processing (ICSLP)*.

Yilmaz, Emine, Peter Hayes, Raza Habib, Jordan Burgess, and David Barber (2021). "Sample Efficient Model Evaluation". In: *arXiv:2109.12043*.

Yoo, Kang Min, Junyeob Kim, Hyuhng Joon Kim, Hyunsoo Cho, Hwiyeol Jo, Sang-Woo Lee, Sang-goo Lee, and Taeuk Kim (2022). "Ground-truth labels matter: A deeper look into input-label demonstrations". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Yoon, Jaesik, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn (2018). "Bayesian Model-Agnostic Meta-Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett.

You, Yang, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh (2020). "Large Batch Optimization for Deep Learning: Training BERT in 76 minutes". In: *International Conference on Learning Representations (ICLR)*.

Yu, Dayou, Weishi Shi, and Qi Yu (2024). "Actively Testing Your Model While It Learns: Realizing Label-Efficient Learning in Practice". In: *Advances in Neural Information Processing Systems*.

Yu, Tong and Hong Zhu (2020). "Hyper-parameter optimization: A review of algorithms and applications". In: *arXiv:2003.05689*.

Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide Residual Networks". In: *British Machine Vision Conference (BMVC)*.

Zeiler, Matthew D (2012). "Adadelta: an adaptive learning rate method". In: *arXiv:1212.5701*.

Zhai, Xiaohua, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer (2022). "Lit: Zero-shot transfer with locked-image text tuning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, Cheng, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt (2018). "Advances in variational inference". In: *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*.

Zhang, Michael, James Lucas, Jimmy Ba, and Geoffrey E Hinton (2019). "Lookahead Optimizer: k steps forward, 1 step back". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhang, Susan, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. (2022a). "Opt: Open pre-trained transformer language models". In: *arXiv:2205.01068*.

Zhang, Xiang, Junbo Jake Zhao, and Yann LeCun (2015). "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhang, Yiming, Shi Feng, and Chenhao Tan (2022b). "Active example selection for in-context learning". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Zhang, Yufeng, Fengzhuo Zhang, Zhuoran Yang, and Zhaoran Wang (2023). "What and How does In-Context Learning Learn? Bayesian Model Averaging, Parameterization, and Generalization". In: *arXiv:2305.19420*.

Zhao, Zihao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh (2021). "Calibrate before use: Improving few-shot performance of language models". In: *International Conference on Machine Learning (ICML)*.

Zhu, Yukun, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler (2015). "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *International Conference on Computer Vision (ICCV)*.

Ziegler, Daniel M, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving (2019). "Fine-tuning language models from human preferences". In: *arXiv:1909.08593*.