Expressivity and Learnability of Sequence Models

Satwik Bhattamishra
Department of Computer Science
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2026

Abstract

In this thesis, we study the representational capabilities and learnability of sequence models, with a focus on Transformers, the backbone of modern large language models. We first investigate *expressivity*: whether and how efficiently an architecture can represent a target function. For a range of algorithmic tasks, we prove exponential separations between Transformers and recurrent/state-space architectures, in the sense that one architecture can compute a function with exponentially smaller model size than the other. We further establish exponential separations between one-layer and two-layer Transformers. We then study *empirical learnability* in stylised in-context learning settings, treating sequence models as learning algorithms. Using a controlled test-bed of Boolean function classes with well-understood learning-theoretic properties, we identify differences in the performance of Transformers, modern recurrent models, and long-convolutional architectures. We further analyse learnability in additional settings, and demonstrate connections between in-context learning in pretrained large language models and the nearest-neighbour algorithm.

We theoretically analyse the learnability of sequence models under richer forms of supervision, focusing on query models where a learner can request labels of desired inputs. We provide new algorithms for learning single-head attention-based models via queries in different oracle models and parameter regimes. We then turn to deterministic finite automata (DFAs) and study their learnability in a new supervision model relevant to the identification of the support of language models. We first characterise the identifiability of DFAs in this setting and establish hardness barriers for learning from examples or from membership queries alone. Finally, we introduce a query model inspired by the information one can obtain from language models and give an efficient learning algorithm by extending Angluin's $L^\star$ algorithm. We validate this approach through a systematic empirical study that extracts DFAs from Transformer-based language models trained on regular languages.

Acknowledgements

I have been incredibly fortunate to have had the support of wonderful mentors, collaborators, and friends throughout my academic journey. First and foremost, I am deeply grateful to my advisors, Varun Kanade and Phil Blunsom, for their guidance and unwavering support during my PhD. They are both brilliant researchers and mentors, and a constant source of inspiration.

From Varun, I learned a great deal about research, learning theory, and how to formalise and approach problems in the right way (among many other things). He has been incredibly patient and generous with his time. One of the aspects of my PhD I will cherish most is the time spent in Varun's office, where we discussed ideas and worked through research problems together. From Phil, I learned a lot about LLM research and, importantly, how to think about the broader picture. Phil also helped me pursue several opportunities, consistently prioritised my interests, and offered advice and perspective that extended well beyond research. During my time at Cohere, I was somewhat surprised that he still managed to find time to write code.

One of my primary motivations for doing a PhD was to have the freedom to work on problems I find interesting and meaningful, and I am grateful that both my advisors gave me ample freedom to pursue the directions I cared about. They nudged me back on track when I wandered off, and supported me through some difficult periods.

Before my PhD, I was also fortunate to learn from several excellent mentors and collaborators. I will remain indebted to Navin Goyal, who taught me the basics of research, supported me in numerous ways, allowed me to work on interesting problems, and emphasised the importance of choosing the right problems and being critical of one's own work—lessons I truly value. I want to thank Partha Talukdar, who gave me my first opportunity to do deep learning research and supported me after my time at his lab. I am grateful to Ashutosh Kumar, my first mentor and collaborator, who taught me many elementary things and was always fun to work with. I want to thank Kabir Ahuja, who was an excellent collaborator and instrumental in several of our projects.

During my PhD, I was fortunate to have outstanding collaborators and friends. I am grateful to Arkil Patel, who has been a great friend, collaborator, and part-time counsellor throughout my PhD. I want to thank Michael Hahn, with whom I've enjoyed working on several projects and had many insightful discussions. I also want to thank Amey Agarwal, who taught me a bit about systems and collaborated with me on a SysML project. I am grateful to Lewis Smith and Siddhartha Kamalakara, from whom I learned a great deal, and with whom I had fun working at Cohere. I want to thank Xingchen Wan, who was very helpful during my internship at Google. I also want to thank Michael Rizvi and Andy Yang, with whom I collaborated on research projects related to Transformers.

My time at Oxford would not have been the same without many people. In particular, I thank Jirko Rubruck, with whom I had a lot of fun climbing, chatting about all sorts of things, and going on trips, among other adventures. I also want to thank Abheek Ghosh, who was a wonderful flatmate, cooked great food, and was always up for bouncing around random ideas. I thank Charles London for our evenings at pubs and conversations about meaningful questions of research and life, among other things. I am grateful to many other friends and colleagues who made my time at Oxford memorable: Timo Geitlinger, Nourah Niazy, Yash Bhalgat, Daniella Ye, Julian D'Costa, Ritesh Agarwal, and Antonia Kormpa.

I am also grateful to Google DeepMind for financially supporting my PhD.

Finally, I am deeply indebted to my parents, Tapan and Minati Bhattamishra, and my brother, Sanket Bhattamishra, for their sacrifices and steadfast support, which allowed me to pursue my interests.

Contents

Chapter 1

Introduction

1.1 Overview and Motivation

Humans have a remarkable ability to learn languages, reason about problems, and generalise in a compositional manner. A long line of work in theoretical computer science and theoretical linguistics has sought to formalise aspects of these abilities. In formal language theory, grammars and state machines are used to precisely describe classes of languages and their structural properties. In learning theory, models such as identification in the limit and PAC learning make precise when a class of functions is learnable from examples or queries. More recently, neural sequence models have evolved rapidly, and researchers have developed models that mimic many aspects of human language processing and reasoning, which now form the basis of widely used systems. Apart from their applicability to various practical problems, they also provide us with a computational object to understand more about learning and reasoning.

To study such systems in a principled way, it is useful to first understand what kinds of tasks neural sequence models can *represent* or *express*, and how costly those representations are in terms of resources such as depth, width, or number of parameters. For instance, one might ask whether there exists a Transformer network that can perform addition, or decide whether two strings are equal, and if so, what the smallest such network is. Questions of this form fall under *expressivity*. Understanding expressivity helps clarify which tasks are naturally easy or difficult for a given sequence-modelling architecture, and where different architectures fundamentally diverge in their capabilities. Moreover, since a model cannot learn what it cannot represent, expressivity results provide an essential prerequisite for reasoning about learning.

At the same time, expressivity does not directly imply *learnability*. The existence of a small model that computes a task does not mean that standard training methods can

efficiently find it from data, nor does it determine how many examples are required. A central goal of computational learning theory has been to formalise and study various aspects of the learning phenomenon from a computational perspective. Such notions let us quantify the complexity of learning different tasks: informally, a task is *efficiently learnable* if there is a learning algorithm that, given a polynomial amount of data, runs in polynomial time and returns a hypothesis with low error. Conversely, learning can be considered *hard* if no algorithm can find a good hypothesis within polynomial time or resources. Hardness can arise for several reasons: the amount of available samples may be insufficient to identify an accurate hypothesis (statistical hardness); even when good hypotheses exist and the data is sufficient, the search problem induced by learning may be computationally intractable (computational hardness); or, as noted above, the hypothesis class may simply not contain a good predictor (a representational limitation).

These considerations motivate studying expressivity and learning side by side, and also motivate richer forms of supervision–beyond i.i.d. labelled examples–that can make certain problems tractable. With this computational viewpoint in place, we now turn to the dominant families of neural architectures for sequence modelling and the different ways they store and process information over time.

Over the past decade, neural sequence models have become the standard approach for modelling text and other sequential data. Transformer architectures [161] are the backbone of large language models (LLMs) [28] and have also been applied to sequences in vision and biology. Earlier recurrent architectures, such as RNNs and LSTMs [73], were central in the development of sequence modelling but are not as parallelisable as Transformers and do not scale well. More recently, there has been considerable interest in developing more efficient recurrent and state-space models [67, 66, 116, 122], which aim to retain some of the advantages of recurrence while mitigating the quadratic inference cost of standard self-attention.

These architectures process sequences in different ways. For an input of length N, a Transformer maintains N hidden vectors and allows each position to attend to all others in parallel. Recurrent and state-space models instead propagate information through a fixed-size hidden state that is updated sequentially. As a result, they impose different constraints on how information can be stored and manipulated over time.

Despite this rapid empirical progress, our theoretical and conceptual understanding of modern sequence models remains comparatively underdeveloped. We still lack a clear picture of what kinds of computations these architectures can implement efficiently, how design choices such as attention versus recurrence constrain their behaviour, and

under what forms of supervision they can learn particular tasks. A more principled understanding is important for several reasons: (i) *Architecture design.* Understanding which tasks are naturally aligned with a given model class, where its limitations lie, and how we can improve them. *(ii) Reliability and Safety.* By enabling formal guarantees and sharper analyses of failure modes. (iii) *Language and learning.* Since neural sequence models can mimic aspects of human language processing and reasoning, they offer concrete, analysable testbeds for hypotheses about language learning and generalisation as well.

1.2 Contributions and Outline

In this thesis, we focus on understanding various questions around the representational capabilities and learnability of sequence models. The main focus is on Transformers since they are currently the most dominant architecture, but we also explore recurrent architectures, as well as consider deterministic finite automata (DFAs). Our goal is to understand both *expressivity*, in the sense of which functions or tasks these models can in principle represent with bounded resources, and *learnability*, in the sense of which such functions can be efficiently learned from data or queries under various learning setups. We try to understand these questions theoretically as well as through rigorous empirical analysis.

Broadly, the first part of the thesis focuses on analysis of the representational capabilities of Transformers and recurrent models. The next part focuses on the empirical abilities of neural sequence models to not only act as classifiers or language recognisers but also act as learning algorithms. The third part focuses on theoretical aspects of learning Transformers and automata under different forms of supervision.

The work in the thesis is an effort to address the following overarching questions:

(i) Expressivity. What kinds of functions and formal languages can Transformers and recurrent/state-space models represent efficiently? Are there natural tasks for which one architecture admits substantially more compact representations than another?

(ii) Empirical learnability. Beyond representational capacity, how do different sequence models compare in their ability to *learn* these functions from data, both as sequence classifiers and as in-context learning algorithms?

(iii) Provable learnability with rich supervision. Can we design efficient algorithms that provably learn sequence models such as DFAs or Transformers when given access to richer forms of supervision, such as blackbox access to their outputs or additional labels from a teacher?

Outline. We begin with a discussion of relevant background on sequential architectures, models of learning, and related work in Chapter 2. The next four chapters form the technical content of the thesis, and their individual contents are described in detail below. We end with a discussion on some implications of the results as well as some open problems in Chapter 7.

Neural sequence models like Transformers and RNNs can be viewed as computational models, much like Boolean circuits or automata, which let us pose familiar complexity-theoretic questions about their capabilities. We say a model can *represent* a function $f : \mathcal{X} \to \{0, 1\}$ if it can produce the correct output for every input in the domain of f. Informally, we will say that a sequence model can represent a family of length-N functions *efficiently* if there exist models whose size grows sublinearly in N (for example, poly-logarithmically in N), and *inefficiently* if any such model must have size linear in N (or larger). We make these notions precise in the context of particular architectures and tasks, and use them to formalise *separation* results: tasks for which one architecture admits efficient representations while another requires much larger resources.

Separations in Expressivity. First, in *Chapter 3*, we analyse the differences in the representational capabilities of Transformers and recurrent models. We begin with a simple task called Index Lookup and show that for sequences of length N, one-layer Transformers of size poly-logarithmic in N can represent the Index Lookup task, whereas no multi-layer RNN or SSM can represent the task with width $o(N)$. The task showcases how we can use communication complexity to derive lower bounds for sequence models and how we can use almost orthogonal vectors to construct small-sized Transformers. Those techniques are then used to derive several further results. Notably, we prove a lower bound for one-layer Transformers on Dyck languages, which leads to a separation between RNNs and one-layer Transformers. Further, we show exponential separations between Transformers and RNNs on tasks like string equality, Boolean functions, and the nearest-neighbour algorithm. Our results also show the first exponential separation between one-layer and two-layer Transformers on Boolean functions such as disjointness and string equality. Lastly, we conduct various experiments to compare the ability of Transformers and RNNs/SSMs to compute tasks like Index Lookup and bounded Dycks, and find that the tasks which are easier for a model to represent generally align with their empirical performance when trained on those tasks. Chapter 3 is based on Bhattamishra et al. [21], which was published at NeurIPS 2024.

Transformers as Learning Algorithms and In-context Learning. Second, in *Chapter 4* we move from worst-case expressivity to empirical learnability, treating Transformers and related architectures as learning algorithms in stylised in-context learning setups. We construct a test-bed of Boolean function classes with well-understood learning-theoretic properties and train Transformers from scratch to perform in-context prediction from sequences of labelled examples. We find that Transformers can nearly match the performance of optimal learning algorithms on several 'simpler' classes, while their performance deteriorates to near-random on more 'complex' classes, such as parities, even though efficient algorithms are known in that case. We also evaluate recently proposed attention-free sequence models and observe that while they often match Transformer's performance, they exhibit clear gaps on some tasks, sharpening our understanding of what aspects of in-context learning are specific to attention. We then introduce *teaching sequences*—carefully chosen examples that uniquely identify a target function—and show that Transformers are able to exploit such sequences to learn more sample-efficiently. In particular, a single Transformer can learn to implement two different algorithms for the same task and adaptively choose the more sample-efficient strategy when the in-context examples form a teaching sequence. Finally, we adapt this framework to pretrained LLMs by (i) training only the input embeddings of a frozen GPT-2 model, and (ii) prompting models such as LLaMA-2 and GPT-4 directly with Boolean inputs. In both cases, we observe non-trivial in-context generalisation competitive with nearest-neighbour baselines on prediction problems that are guaranteed not to be contained in the pretraining data, indicating that these models can implement genuine learning algorithms to some extent over in-context examples. Chapter 4 is based on Bhattamishra et al. [22], which appeared at ICLR 2024.

Learning Attention with Queries. In *Chapter 5*, we focus on provable learnability and present new algorithms for learning attention-based sequence models from blackbox access to their outputs. In this setting, a learner can query any sequence of vectors to a value-query oracle and obtain the output of the target model. We begin with one of the simplest sequence architectures: a single-head softmax attention-based regressor with width d. We show an elementary algorithm that exactly recovers the parameters of such a model using only $O(d^2)$ queries. We then prove that any algorithm for learning one-hidden-layer ReLU feedforward networks can be leveraged, via a simple reduction, to learn one-layer Transformers with single-head attention under the structural assumptions required for the feedforward network. When the head dimension $r \ll d$, we further obtain a randomised algorithm, based on ideas

from compressed sensing, that learns the single-head attention model with only $O(rd)$ queries. We also analyse robustness to noise in the oracle, showing that the parameters can still be estimated to accuracy ϵ in polynomial time when the oracle outputs are only specified up to an additive tolerance τ with $1/\tau = O(d)$. Finally, we consider multi-head attention and show that, in general, the parameters are not identifiable, the parameter–function map is not injective, and hence learnability in the same sense is impossible without further assumptions. Chapter 5 is based on Bhattamishra et al. [24], which is accepted at ICML 2026.

Learning Automata with NSP queries. In *Chapter 6*, we analyse the learnability of deterministic finite automata (DFA) in a new setting tailored towards learning the support of modern language models. In this setting, called *Next Symbol Prediction* (NSP), a learner has access to only positive examples together with, for every prefix, (i) which prefix is in the language, and (ii) which next symbols can lead to an accepting string. This setting has been used in prior work to empirically analyse neural sequence models, and additionally, we observe that efficient algorithms for the NSP setting can be used to learn the (truncated) support of language models.

We first show that the class of DFAs with at most n states is identifiable from positive examples augmented with these NSP labels. Nevertheless, even with this richer supervision, we show that PAC-learning DFAs remains computationally hard, and exact identification using only membership queries cannot be achieved in polynomial time. We then present $\mathrm{L}^{\star}_{\mathrm{nsp}}$, an extension of Angluin's $\mathrm{L}^{\star}$ algorithm, and show that DFAs can be PAC-learned efficiently using a language-model–based teacher that answers membership queries and generates valid strings conditioned on prefix prompts. Finally, we conduct a comprehensive empirical evaluation on 11 regular languages of varying complexity. Using $\mathrm{L}^{\star}_{\mathrm{nsp}}$, we extract DFAs from Transformer-based language models trained on regular languages to evaluate the algorithm's effectiveness and identify erroneous examples. Chapter 6 is based on Bhattamishra et al. [23] which appeared at ICLR 2026.

Chapter 2

Background

2.1 Sequence Models and Architectures

Transformers are part of every technical chapter in the dissertation. Broadly, we follow the standard definition of Transformers [161] but use them in different forms such as classifiers, regressors, or as language models. For technical chapters with proofs related to Transformers, we define the exact model and notation formally in the corresponding chapter for easier readability. Below, we define Transformers in a general manner, followed by a discussion of the specifics of different chapters.

Transformers can be viewed as a sequence model that maps an input matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$ (a length-N sequence of d-dimensional vectors) to an output of the same shape $\mathbb{R}^{N \times d}$. Each layer consists of two blocks applied to $\mathbf{X}$: a multi-head self-attention block and a position-wise feedforward block. Layers are stacked, and in practice, both blocks are wrapped with residual connections and layer normalisation.

Self-attention. For a single attention head with head dimension d_{head}, we fix projection matrices $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d_{\text{head}} \times d}$ and define the query, key, and value matrices $Q(\mathbf{X}) = \mathbf{X}\boldsymbol{W}_Q^\top$, $K(\mathbf{X}) = \mathbf{X}\boldsymbol{W}_K^\top$, and $V(\mathbf{X}) = \mathbf{X}\boldsymbol{W}_V^\top \in \mathbb{R}^{N \times d_{\text{head}}}$. Writing $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ for the i-th rows of these matrices, the attention weights and output at position i are

$$\alpha_{ij}(\mathbf{X}) = \frac{\exp(\langle \mathbf{q}_i, \mathbf{k}_j \rangle)}{\sum_{\ell=1}^{N} \exp(\langle \mathbf{q}_i, \mathbf{k}_\ell \rangle)}, \qquad \mathrm{Att}(\mathbf{X})_i = \sum_{j=1}^{N} \alpha_{ij}(\mathbf{X})\, \mathbf{v}_j.$$

Collecting the rows gives the single-head attention output $\mathrm{Att}(\mathbf{X}) \in \mathbb{R}^{N \times d_{\text{head}}}$.

Multi-head attention. Multi-head attention uses H heads in parallel. Head $h \in [H]$ has its own projection matrices $\boldsymbol{W}_Q^{(h)}, \boldsymbol{W}_K^{(h)}, \boldsymbol{W}_V^{(h)}$ and produces an output $\mathrm{Att}^{(h)}(\mathbf{X}) \in \mathbb{R}^{N \times d_{\text{head}}}$. These are concatenated along the feature dimension and then

projected back to $\mathbb{R}^{N\times d}$ using an output matrix $\boldsymbol{W}_O \in \mathbb{R}^{d\times(Hd_{\text{head}})}$. We denote the resulting map by $\text{M-Att}_H : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$.

Feedforward block. The feedforward block applies the same non-linear transformation to each position independently. Concretely, it is implemented as a small multilayer perceptron $F_{\text{ff}} : \mathbb{R}^d \to \mathbb{R}^d$ with a generic activation function (e.g. ReLU or GELU). Given $\mathbf{Y} \in \mathbb{R}^{N\times d}$, the feedforward block produces an output in $\mathbb{R}^{N\times d}$ by applying F_{ff} to each row of $\mathbf{Y}$.

Residual connections and layer normalisation. A standard Transformer layer combines these components using residual connections and layer normalisation. Starting from $\mathbf{X}^{(\ell)} \in \mathbb{R}^{N\times d}$, the layer first applies layer normalisation, then multi-head attention, and adds the result back to $\mathbf{X}^{(\ell)}$ (a residual connection). A second layer normalisation and the feedforward block are then applied, again with a residual connection.

From symbols to vectors. For language and other discrete sequence tasks, the model operates on a vocabulary Σ. An input $w_1 \cdots w_N \in \Sigma^*$ is converted into a sequence of vectors by an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{|\Sigma|\times d}$ together with positional encodings. The i-th token w_i is mapped to an embedding $\mathbf{x}_i \in \mathbb{R}^d$, and these vectors form $\mathbf{X} \in \mathbb{R}^{N\times d}$, which is then processed by the Transformer layers to produce a contextualised representation $\mathbf{Z} \in \mathbb{R}^{N\times d}$.

Language modelling. In an auto-regressive language model, attention is restricted by a causal mask so that position i can only attend to positions $1, \ldots, i$. Given the final representation $\mathbf{z}_i$ at position i, a linear projection $\boldsymbol{W}_{\text{LM}} \in \mathbb{R}^{|\Sigma|\times d}$ produces logits over the vocabulary, and a softmax over symbols in Σ defines the conditional distribution $p(w_{i+1} \mid w_{1:i})$. Thus the Transformer, together with this output layer, defines a language model on Σ^*, while internally still implementing a map $\mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$.

Classification and regression. For sequence-level prediction, the final sequence $\mathbf{Z}$ is reduced to a single vector, for example by taking the representation of a special "[CLS]" token or by taking the output vector at the last position. A linear map $\boldsymbol{W}_{\text{out}} \in \mathbb{R}^{k\times d}$ followed by a suitable activation then yields either a k-class classifier (with a softmax over classes) or a real-valued regressor (with, for instance, an identity or scalar nonlinearity on the output).

Pointers to later chapters. For the experiments in all the chapters, we use implementations of Transformers based on the definition above. They are used as language models in Chapters 4 and 6, and the implementation follows closely to how they are used in practice. In Chapter 3, Transformers are primarily viewed as sequence classifiers. For our constructions or upper bounds on Transformers, we use

the ReLU activation function and consider the model without residual connections and layernorm — so the bound applies to a slightly simpler model. For our lower bounds, we allow the mapping after the attention block to be arbitrary, and hence they apply to all activation functions as well as with components such as layernorm and residual connections. Chapter 5 primarily views Transformers as a regression model that outputs a scalar.

Recurrent and state-space models. Recurrent models process a sequence one element at a time by maintaining a hidden *state* vector that is updated iteratively as new inputs arrive. Formally, given an input sequence $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with $\mathbf{x}_t \in \mathbb{R}^d$, a recurrent or state-space model sequentially updates $\boldsymbol{h}_t$ based on the previous hidden state $\boldsymbol{h}_{t-1}$ and the current input $\boldsymbol{x}_t$. In general form, recurrent models can be formulated as:

$$\mathbf{h}_t = G_{(t)}(\mathbf{x}_t, \mathbf{h}_{t-1}), \qquad y_t = F_{(t)}(\mathbf{h}_t), \quad t = 1, \ldots, N.$$

$G_{(t)}$ is an arbitrary state-transition function, and $F_{(t)}$ is the output function. The state dimension (say $\mathbf{h}_t \in \mathbb{R}^m$) plays the role of the *width* of the model. In practice, when we operate over finite precision p-bit numbers, the hidden state can be viewed as an mp-bit memory, and we refer to mp as the *representation size*.

This abstract formulation subsumes a wide range of architectures. In an Elman RNN, the transition is given by a linear transformation followed by a nonlinearity, e.g. $\boldsymbol{h}_t = \sigma(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t)$ for suitable matrices and activation σ, and the output y_t is obtained by an additional linear map (optionally followed by a nonlinearity). LSTMs and GRUs modify the transition function with gating mechanisms that allow the model to control how information is stored and forgotten. While such non-linear RNNs are expressive, their sequential dependence on $\mathbf{h}_{t-1}$ makes them difficult to parallelise across time during training.

To address this, a large body of work has focused on *linear RNNs* and *state-space models (SSMs)*, in which the transition $G_{(t)}$ has a structured, often linear form. For example, one can take $\mathbf{h}_t$ to evolve according to a linear recurrence driven by $\mathbf{x}_t$, so that the entire sequence of states can be expressed as convolutions of the input with learned filters. This structure allows efficient parallelisation over time using FFT. Architectures such as DSS [67], and Mamba [66] can be viewed as particular instantiations of this general state-space template, with carefully chosen transition structure and parameterisation. Additionally, linear Transformers with causal masking can also be viewed as linear recurrent models [82].

Pointers to later chapters. The exact definitions of different instantiations of recurrent or state-space models are not required to understand the results in the dissertation. Our lower bounds in Chapter 3 apply to the general form described above and thus apply to most, if not all, practical instantiations of RNNs. In our experiments in Chapter 3 and 4, we evaluate and compare many instantiations of RNNs/SSMs mentioned above. For their exact definitions, please refer to the original papers.

Finite-state automata. We will also consider classical models of computation with a finite amount of memory. A deterministic finite automaton (DFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F_A)$ where Q is a finite set of states, Σ is a finite input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F_A \subseteq Q$ is the set of accepting states. Given an input string $w_1 \cdots w_N \in \Sigma^*$, the automaton processes the symbols sequentially by updating its state as $q_t = \delta(q_{t-1}, w_t)$ starting from q_0. The string is accepted if the final state q_N lies in F_A.

Pointers to later chapters. Chapter 6 will study the learnability of DFAs in a new setting and explore the extraction of DFAs from language models. As is natural to observe, DFAs and RNNs are closely related. One can also obtain lower bounds on the size of the hidden state of RNNs/SSMs based on the number of states required by the minimum-sized DFA to compute a function. Such lower bounds apply to the general class of RNNs where the transition function $G_{(t)}$ is fixed across time. Lower bounds based on communication complexity in Chapter 3 apply to slightly more general versions of RNNs where the transition map $G_{(t)}$ is allowed to vary across time.

2.2 Models of Learning

The concept of learning has been formalised in many different ways in the field of computational learning theory, depending on what kind of examples, as well as additional supervision, are accessible to a learner. Here, we describe a few models of learning that are relevant to the dissertation.

Throughout, the setting assumes that there is a class of (possibly infinite) functions $\mathcal{F}$ and an unknown target labelling function $f^\star \in \mathcal{F}$. Informally, a learner may have access to labelled examples $(x, f^\star(x))$ or could possibly request the labels of desired examples. The goal of the learner is to either identify $f^\star$ or to find a function $\hat{f}$ such that it can accurately predict the labels of unseen examples measured by some loss or

error function $\text{err}(\hat{f}; f, \mathcal{D})$. The error function is typically the mean-squared error or the cross-entropy error, and could be distribution-dependent depending on the setting.

PAC learning. A canonical way of studying learnability from random examples is Valiant's [160] model of probably approximately correct (PAC) learning. We say a learning algorithm $\mathcal{A}$ is an efficient PAC-learning algorithm for a class $\mathcal{F}$ if for any unknown target labelling function $f^\star \in \mathcal{F}$, and any distribution $\mathcal{D}$, the algorithm can take a polynomial number of labelled examples from $\mathcal{D}$, and produce a hypothesis h in polynomial time such that with probability at least $1 - \delta$, its error $\text{err}(h; f^\star, \mathcal{D}) \leq \epsilon$.

Membership and Value Queries. Instead of access to random examples, another model of learning involves access to a membership or value query oracle. In this setting, for any unknown target function $f^\star$, a learner can query the label of any input x and obtain $f^\star(x)$. If the class of functions contains binary classifiers, i.e., $f^\star : \mathcal{X} \to \{0, 1\}$, then we refer to the oracle as a membership query oracle $\text{MQ}(x) = f^\star(x)$. If the class of functions is over reals, such as regression problems, $f^\star : \mathcal{X} \to \mathbb{R}$, then we refer to the oracle as a value query oracle, $\text{VQ}(x) = f^\star(x)$.

Equivalence queries. Another model of learning involves access to an equivalence query oracle EQ, where the learner can query the oracle with a hypothesis function, and the EQ oracle returns one of two things. If the hypothesis function is the same as the target function then the oracle returns TRUE, and if $\hat{f} \neq f^\star$, then the EQ oracle must return a counterexample $x \in \mathcal{X}$ such that $\hat{f}(x) \neq f^\star(x)$. Learnability with equivalence queries focuses on exact learning, unlike PAC-learning. It is known that if there is an exact learning algorithm for a class of functions with equivalence queries only, then it implies that there is a PAC-learning algorithm, but the converse is not necessarily true.

Pointers to later chapters. In Chapter 6, we study learnability of automata in most of the settings described above, such as PAC-learning, learning with membership queries, equivalence queries, as well as a new oracle called generative queries. Chapter 5 focuses on the learnability of attention-based models with value queries and their variants. In Chapter 4, we will explore the ability of neural sequence models to mimic learning algorithms, and our empirical analysis is designed with the viewpoint of PAC-learning in mind.

2.3 Historical Context and Related Work

2.3.1 Computational Capabilities of Neural Sequence Models

Early Works. Analysis of the computational capabilities of neural sequence models was an active area in the 1980s and 1990s [87, 128]. A number of works studied the expressivity of earlier variants of recurrent models [143, 60] and attempted to understand the conditions under which they are Turing complete. Notably, under idealised assumptions, Siegelmann and Sontag [142] showed that finite-sized recurrent networks can simulate Turing machines and are hence Turing-complete. Early works also connected recurrent networks to automata-theoretic perspectives and formal language recognition, helping frame sequence models as computational devices rather than purely statistical predictors [42, 98].

Relation to language and algorithmic behaviour. Formal languages are abstract models of the syntax of programming and natural languages; they also relate to cognitive linguistics, e.g., see Jäger and Rogers [76] and references therein. Formal languages provide controlled settings for the kinds of structure that matter in natural and programming languages [151, 136, 139]. They also connect to long-standing linguistic questions about which computational classes best describe human language. The seminal work of Chomsky [39] argued that the presence of hierarchical dependencies makes DFAs incapable of modelling natural language, which leads to the conjecture that natural languages are at least context-free. Later work by Shieber [141] showed that some languages, such as Swiss-German, contain cross-serial dependencies that are beyond context-free. Hence, natural language is considered to be a mildly context-sensitive language, which is a superset of context-free languages but a subset of context-sensitive languages. Some components of language (and animal communication) appear compatible with regular or subregular descriptions [17, 76]. Complementary to their ability to capture structural properties of natural languages, formal languages also provide us with a tool to understand the kind of algorithmic behaviour and programming languages [147, 2] that a neural network can model.

Expressivity of Recurrent Models. Over the last decade, a strand of work has revisited such questions about modern variants of recurrent sequence models. It is known that in finite precision (practical setting) the expressive power of RNNs is equivalent to that of a DFA [65, 88]. In practice, recurrent models such as LSTMs and GRUs are effective at learning various regular languages [12, 18]. Beyond regular languages, recurrent models have been extensively evaluated on context-free languages involving hierarchical dependencies using Dyck-style benchmarks and related setups

[151, 145, 177, 72], with memory-augmented variants (e.g., stack-like mechanisms) often improving out-of-distribution generalisation [152]. While certain recurrent models can express deterministic context-free languages with infinite precision [88, 19], in practice, they struggle to generalise to hierarchical dependencies with higher depths [19, 152]. More recently, connections between LSTMs and counter automata have been established empirically [168, 150] and theoretically [102].

Expressivity of Transformers. With Transformers becoming the dominant sequence model, many works have sought principled characterisations of their expressive power [e.g. 68, 135, 148]. In the limit of unbounded resources, Transformers are universal function approximators [178] and can be Turing complete [124, 20]. Under bounded precision and related constraints, their power has been linked to circuit classes and parallel computation [71, 103, 101] as well as to logical formalisms [38]. At the same time, even when a language is representable in principle for bounded lengths, empirical studies have repeatedly found that Transformers can struggle to *generalise* on algorithmic tasks that require modular counting [18, 37, 48, 55]. Importantly, much of this literature targets asymptotic expressivity of a fixed-sized architecture as input length grows. Our study in Chapter 3 instead aims for a more fine-grained and practically grounded view by analysing how the required network size must scale with the input. The closest to our work is Sanford et al. [134], who introduced communication-complexity lower bounds for Transformers and RNNs on abstract detection and sparse-averaging tasks; among other things, we extend this line to separations on natural tasks, prove slightly more general results, and show depth separation results.

Automata extraction from neural models was introduced by Giles et al. [60], Omlin and Giles [113] and has been an active field [163, 110]. Broadly, the goal of this line of work is to extract interpretable computational objects, such as DFAs, from blackbox models, such as neural networks, to better understand their mechanics and robustness. Notably, Weiss et al. [167] developed a method for white-box extraction based on the $L^\star$ algorithm [7], and several subsequent works have focused on the extraction of weighted automata/PDFAs [169, 166, 53]. Recent work [179] has also explored $L^\star$-like methods for Transformer-based classifiers. Mayr et al. [100] learn a PDFA abstraction of an LM via a distributional congruence. In Chapter 6, we focus on support identification of neural language models, where we characterise learnability and conduct empirical analysis based on a new provable extension of the $L^\star$ algorithm.

2.3.2 In-Context Learning

Stylised settings. In-context learning is an empirical phenomenon where LLMs could evidently learn to make predictions for new tasks after being prompted with a few labelled examples as inputs. Understanding the in-context learning (ICL) phenomenon has attracted significant interest in recent years. Since the ICL phenomenon was demonstrated in Brown et al. [28], numerous works have investigated the practical capabilities and limitations of LLMs to perform ICL (see Dong et al. [50] for a survey). Since it is infeasible to define real-world data and tasks precisely, several works have studied it in stylised, well-defined settings [32, 69, 173]. Garg et al. [57] presented a meta-learning-like framework and demonstrated that Transformers can match gradient-based learning algorithms for regression tasks. Subsequent works [44, 4, 162] demonstrated that Transformers were expressive enough to represent gradient-descent and empirically verified that the learned weights of linear Transformers indeed computed gradient-based algorithms. Li et al. [92], Bai et al. [14] studied the sample complexity of in-context learning linear regression and other real-valued functions with Transformers. More recently, [119, 14] explored (among other things) the efficacy of Transformers in learning mixtures of tasks in-context and showed that they could adaptively select appropriate algorithms for a given sequence of inputs and labels.

Realistic settings. In recent years, there have been many empirical works [94, 107, 96, 132, 106, 112, 165] that have analysed the capabilities and limitations of in-context learning (ICL) in large language models (LLMs). Liu et al. [94] explored various strategies for selecting examples that improve in-context learning performance. Lu et al. [96] demonstrated that the in-context learning ability of models is also sensitive to the order in which the samples are provided in the prompt. Razeghi et al. [132] worked with numerical reasoning tasks and showed that in-context learning performance is stronger for instances whose terms are more prevalent in the training data. Min et al. [106] questioned whether models really learned new tasks in-context by showing strong ICL performance even when the prompt labels are chosen randomly. [112] and [52] frame the in-context learning ability slightly differently by referring to any model that gets better (i.e., shows a decrease in loss) with more context (i.e., increasing token indices) as an in-context learner. They provide arguments for the existence of special circuits, which predict the next token by observing similar patterns previously seen in context, inside the Transformer model, that are responsible for in-context learning. More recently, Wei et al. [165] showed that the ability of models to override semantic priors and effectively learn the task presented in-context emerges with scale.

2.3.3 Learnability of Sequence Models

Early foundational works in learning theory sought to formalise learning from an algorithmic point of view. A notable work by Gold [62] introduced identification in the limit setting, where a learner receives an infinite sequence of positive examples and is required to eventually converge to the target language. Gold [63] showed that certain classes, such as regular languages, are not identifiable from positive examples only. Angluin [6] provided a clearer characterisation of which formal languages are learnable from positive examples in the identification in the limit setting. Around the same time, Valiant [160] introduced the Probably Approximately Correct (PAC) learning framework, which grounded learnability in terms of statistical guarantees (e.g., how many labelled examples suffice to achieve low error) and computational efficiency (whether a learner can run in time polynomial in the relevant parameters). In Valiant's model, some concept classes admit efficient PAC learners, while for others, efficient PAC learning is believed to be computationally intractable under standard complexity-theoretic or cryptographic assumptions [125, 84]. Beyond learning from examples, researchers studied learnability with different forms of teachers and query models [8, 29].

Learnability of Regular Languages. Several works showed that learning DFAs from labelled examples is computationally hard – finding the minimum consistent DFA is NP-hard [63, 5, 126], and even representation-independent (improper) PAC learning is intractable under cryptographic assumptions [84]. This motivated stronger models of learning, such as including queries where a learner could adaptively request the label of any desired input. The seminal work of Angluin [7] showed that with access to both random examples and membership queries, regular languages are efficiently learnable. Under structural assumptions, several works have studied PAC learnability for probabilistic DFAs [41, 118]. The area remains active with recent theory on counterexample handling and lower bounds in the MAT framework [159, 89]. We build on this line of research by characterising the learnability of DFAs in a new setting relevant to learning the support of language models, and by situating prior empirical analyses of neural nets in the appropriate context.

Learning Neural Nets with Queries. Learning with membership/value queries became a standard topic in computational learning theory [8, 29], and it is also a natural formalisation of black-box *model extraction* where an adversary adaptively probes a prediction API. Early work investigated whether a feedforward network can be reconstructed from oracle access to its input–output map [54]. More recently, empirical methods have been proposed for extraction attacks and have been studied

extensively in the context of security and ML [158, 115, 77]. On the theory side, there are now polynomial-time guarantees for learning shallow ReLU networks under different query models and structural assumptions, including value-query learning under Gaussian inputs [35], linear independence of parameter vectors [105], and exact parameter extraction under general-position conditions [45]. In contrast, despite the central role of attention in modern sequence models, we are not aware of prior work giving provable learnability/parameter estimation guarantees for softmax attention (or Transformers built from it) from queries; our results in Chapter 5 initiate this study for attention blocks. Complementary to our work, Carlini et al. [31] tackle the problem of extracting specific parts of a production-level language model, where they recover limited information such as width and embedding matrices but in a more challenging and realistic setting.

Learning Neural Nets and Transformers from Examples. Analysing deep learning models and establishing provable guarantees with minimal assumptions for deep networks has remained quite challenging [137, 146]. While deep learning models work well in practice, establishing efficient PAC learning guarantees has been difficult, with several works showing that efficient PAC learning is likely to be hard under standard complexity-theoretic/cryptographic assumptions [27, 61, 91]. This mismatch is also partly due to the hardness results indicating difficulty in the worst-case sense when considering all data distributions. To mitigate this and understand deep learning models, researchers have focused on analysing shallow networks under distributional and structural assumptions [e.g. 58, 36, 49].

Similarly, to better understand the learning dynamics of Transformers, there is a growing body of work that studies the learnability of a single-layer attention-based model under various assumptions [e.g. 34, 154, 97, 99]. For instance, building on [134], Wang et al. [164] analyse the learnability of sparse token selection with a single-head attention model and prove separations from fully-connected nets. On the optimisation side, Arnaboldi et al. [9] study training dynamics for simplified attention-style models, and recent high-dimensional/statistical-mechanics analyses quantify when softmax attention detects weak sparse signals and how it compares to linear alternatives [99, 15]. Complementarily, several works characterise implicit bias and geometry in attention training, including max-margin/token-selection viewpoints and connections to SVMs [11, 153, 138], and Huang et al. [75] study learning dynamics for in-context learning with one-layer softmax Transformers. Finally, when softmax is removed (linear attention), Yau et al. [176] show PAC learnability via a reduction to kernel methods. These works focus on passive learning and optimisation from random

examples, whereas in Chapter 5, we study efficient learnability and parameter recovery of softmax-attention with black-box value queries.

Chapter 3

Separations in the Representation Capabilities of Transformers and RNNs

3.1 Introduction

Transformers [161] are the go-to architecture for building LLMs [28], but recently, there has been significant interest in reviving recurrent architectures to build LLMs for practical tasks [66, 116, 122] to circumvent the quadratic complexity of inference for Transformers. While Transformers process all tokens in parallel, maintaining N vectors, and are in some sense stateless, recurrent models primarily store information in a fixed-size hidden state that they can update during the course of the computation. This raises a fundamental question that we explore: Are there tasks that are computationally easier for one architecture to represent but significantly more difficult for the other one?

Ever since Transformers supplanted LSTMs, there has been significant interest in understanding the differences in the computational capabilities of the models both theoretically and empirically. On the one hand, even as Transformers have been widely adopted for building LLMs, several studies found that they struggled at modelling various formal languages, particularly those that required modular counting or state-tracking [18, 48, 93]. At the same time, despite substantial effort, it has proven hard to match the performance of Transformer-based LLMs at scale with recurrent architectures [46, 122, 66]; in particular, it has been observed that LLMs based on recurrent models struggle on associative recall or extraction-related tasks [46, 10]. More recently, Arora et al. [10] demonstrated that Transformers are better than attention-free models at synthetic tasks based on associative recall. Based on

these observations, it is natural to wonder if such tasks are theoretically easier for Transformers to represent in comparison to recurrent models.

Our Contributions. We show differences in the representational capabilities of Transformers and recurrent models across several natural tasks of real-world relevance. In this chapter, we show strong *separation* results: when a task is easy, we show that it can be expressed by one architecture with size poly-logarithmic in the input length N; on the other hand, we show the other type of architecture requires the size to be linear in N. We describe the tasks studied and our key results below.

(i) Index Lookup. Given a sequence of symbols $s_1, \ldots, s_N$ followed by a position $p \in [N]$, a model has to output the symbol in the p-th position s_p (Figure 3.1a). Our first result shows that one-layer Transformers with size poly-logarithmic in N can express this task (Theorem 1) whereas any recurrent model must have width $\Omega(N)$ to perform this task (Theorem 3).

(ii) Bounded Dycks. Dyck languages with bounded depth require a model to recognise whether a string of parentheses is well-balanced (Figure 3.1b). These formal languages have received a great deal of attention in the study of neural sequence models because they capture hierarchical dependencies that occur in natural languages [51, 72, 68, 19]. They are also central to formal language theory as all context-free languages can be expressed in terms of Dyck languages [40]. In contrast to the results for index lookup, we show that one-layer Transformers must have a size that grows linearly in input length to represent this task (Theorem 5), whereas prior works found that constant-size recurrent models can express this task [72, 19].

(iii) String Equality. This task is formalised by the Equality function $\mathrm{EQ}(\mathbf{x}, \mathbf{y}) = \mathbb{I}[\mathbf{x} = \mathbf{y}]$ where $\mathbf{x}, \mathbf{y}$ are two strings of length N; it models the natural task of matching two documents. We show that log-sized two-layer Transformers can represent the function EQ (Theorem 6), whereas both one-layer Transformers and recurrent models must have a size that grows linearly with the sequence length. These results extend to a broader class of Boolean functions.

(iv) Nearest Neighbour and Associative Recall. In this task, a model is provided with a sequence of inputs and labels $\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}$ followed by a query input $\mathbf{x}_k$. The model has to determine the label y_k of the query input by applying the nearest neighbour algorithm in its forward pass (Figure 3.1c). This task subsumes various associative recall tasks considered in earlier works (cf. Section 3.5.2). We show that a two-layer Transformer with size logarithmic in the input length can perform this task (Theorem 7), whereas recurrent models require a size linear in the length (Theorem 8).

We also empirically investigate the performance of Transformers and standard recurrent models [73], including recently proposed state-space models [67, 66] on these tasks. The observed behaviour is along the lines indicated by our theoretical results.

Our Techniques. For constructing Transformers that perform the tasks, one of the key requirements is the ability to precisely attend to specific positions. In order to do this with low-dimensional embeddings, our constructions make use of nearly orthogonal vectors obtained using the Johnson-Lindenstrauss lemma [80]; furthermore, these can be generated efficiently in logarithmic space, which allows the size of the models to be poly-logarithmic in the input length (cf. Section 3.2.3).

For our lower bounds, we appeal to results from communication complexity. We show how to obtain protocols with communication complexity bounded by the size of the models. Together with established (and new) communication complexity lower bounds, we obtain lower bounds on the model size. For our lower bound for one-layer Transformers, we derive a lower bound on the communication complexity for bounded Dyck languages (Lemma 3).

3.2 Definitions and Preliminaries

We consider two types of models: Transformers [161] and recurrent models. We use recurrent models to refer more generally to nonlinear RNNs such as LSTMs [73], state-space models [67, 66] as well as variants of linear Transformer [82, 149] which can process inputs in a recurrent manner [82].

For some finite alphabet Σ, a sequence model is given a sequence in Σ^N and depending on the task outputs either $\{0, 1\}$ or a sequence of outputs. Each $s_i \in \Sigma$ from a sequence $s_1 \cdots s_N \in \Sigma^N$ is mapped to a vector in $\mathbf{R}^d$ via an embedding map $\phi : \Sigma \to \mathbf{R}^d$. For Transformers, the input embedding function further takes the position i as input, along with s_i. Each layer for Transformers and recurrent models maps inputs from $\mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$. A model M has *fixed precision* p if all the parameters of the model, as well as the values in the intermediate vectors, can be implemented with p-bit precision numbers.

Transformers. Each layer of a Transformer has an attention block followed by an MLP block. The attention block takes as input $\mathbf{X} \in \mathbb{R}^{N\times d}$ and applies the operation $\mathrm{Att}(\mathbf{X}) = \mathrm{softmax}(\mathbf{X}\mathbf{W}_Q^\top \mathbf{W}_K \mathbf{X}^\top)\mathbf{X}\mathbf{W}_V^\top$ where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{m\times d}$. For simplicity, we will use $Q(\mathbf{x}_i)$ (and likewise $K(\mathbf{x}_i)$ and $V(\mathbf{x}_i)$) to denote $\mathbf{W}_Q\mathbf{x}_i$. The *width* of the Transformer is $\max(m, d)$, where $m \times d$ is the shape of the projection matrices $\mathbf{W}_Q, \mathbf{W}_K$. For any matrix $\mathbf{A} \in \mathbb{R}^{N\times M}$, the softmax operator is applied row-wise

as follows $\text{softmax}(\mathbf{A})_{i,j} = \frac{\exp(\mathbf{A}_{i,j})}{\sum_{k=1}^{M}\exp(\mathbf{A}_{i,k})}$. Multi-head attention with H heads is defined as $\text{M-Att}_H(\mathbf{X}) = [\text{Att}_1(\mathbf{X}), \ldots, \text{Att}_H(\mathbf{X})]\mathbf{W}_O$ where each $\text{Att}_i(\mathbf{X})$ has its own set of parameters. The matrix $\mathbf{W}_O \in \mathbb{R}^{mH\times d}$ projects the concatenated vector to a vector of dimension d.

For an input $\mathbf{X} \in \mathbb{R}^{N\times d}$, the output of a layer of Transformer will be $\psi(\text{M-Att}(\mathbf{X})) \in \mathbb{R}^{N\times d}$ where $\psi : \mathbb{R}^d \to \mathbb{R}^d$ is a feedforward network. We use $\text{TF}^L_{m,p,H}$ to denote the class of all Transformers operating over p-bit precision numbers with width m, H heads, and at most L layers.

Recurrent Models. A general recurrent neural network (RNN) takes as input the sequence $\mathbf{x}_1, \ldots, \mathbf{x}_N$ where $\mathbf{x}_i \in \mathbb{R}^d$ and produces an output sequence $y_1, \ldots, y_N$; in this chapter we will mostly consider the case when $y_i \in \{0, 1\}$. An RNN with a hidden state of size m over p-bit numbers can be defined as follows. The hidden state is an mp-bit memory $\mathbf{h}_i \in \{0, 1\}^{mp}$ and for some $\mathbf{h}_0 \in \{0, 1\}^{mp}$, the RNN computes $\mathbf{h}_t = g_{(t)}(\mathbf{x}_t, \mathbf{h}_{t-1})$ and $y_t = f_{(t)}(\mathbf{h}_t)$ for $t = 1, \ldots, N$, and $g_{(t)}$ and $f_{(t)}$ are arbitrary functions. Since the transition function is allowed to be arbitrary, this definition captures the general family of recurrent or state-space architectures including LSTMs, state-space models, and linear Transformers, each of which differs in the way the transition function is defined. For a recurrent model, we say that the *representation size* of a hidden state is mp, and its *width* is m, i.e. the hidden state consists of m units each of which uses p-bits. Throughout the chapter, we will use recurrent models or RNNs to refer to the general family of recurrent architectures mentioned above.

By the *representation size* of a model, we will refer to the total number of bits required to represent the model, including all the parameters and embeddings. For Transformers, this is $\Theta(mdpH)$. For a recurrent model, the representation size is at least mp.

3.2.1 Communication Complexity

Our lower bounds for recurrent models and 1-layer Transformers are based on communication-complexity bounds. We assume some familiarity with communication complexity (see Rao and Yehudayoff [130], Kushilevitz and Nisan [90] for an introduction). We will primarily focus on the two-party setting where the communication complexity of a function indicates the number of bits two parties must exchange in order to compute the output of a function. If Alice and Bob want to compute a function $f : \{0, 1\}^N \to \{0, 1\}$ where Alice has the bits in the indices $I \subset [N]$ and Bob has the bits in indices $[N] \setminus I$, the communication complexity of f over that partition is

the minimum number of bits they must exchange to compute the output for all inputs $x \in \{0,1\}^N$. Alice and Bob are allowed unbounded computational power. If Alice and Bob must exchange at least k bits to compute a function $f(x)\, \forall x \in \{0,1\}^n$ over any partition of the input then we say that the communication complexity $C(f) \geq k$. If they use a communication protocol where only one party is allowed to send bits to the other party, then it is called one-way communication, and the communication complexity of the function in that setting is referred to as one-way communication complexity.

For our results, we will use the following three well-known facts about the communication complexity of the Disjointness, Equality, and the INDEX problem.

Disjointness. The disjointness function takes two sets $A, B \subseteq [N]$ as input and returns 0 if the two sets are disjoint and returns 1 otherwise. This can also be seen as a function $\textsc{Disj} : \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$ over two Boolean inputs such that $\textsc{Disj}(a,b) = \max a_i b_i = \mathbb{I}[a^T b > 0]$. If Alice has the input vector $a \in \{0,1\}^N$ and Bob has the vector $b \in \{0,1\}^N$, then the communication complexity of DISJ indicates the minimum number of bits that Alice and Bob will have to exchange to determine the output of the DISJ function. The following is a well-known fact about the disjointness problem,

Fact 1. *[Disjointness [174]] If Alice and Bob have two inputs $a, b \in \{0,1\}^N$, then any deterministic communication protocol used by them to compute $\textsc{Disj}(a,b) = \max a_i b_i$ must exchange at least N bits. Moreover, the randomised communication complexity of the DISJ is $\Omega(N)$.*

Fact 2. *[Equality [130, Ch. 1]] If Alice and Bob have two inputs $a, b \in \{0,1\}^N$, then any deterministic communication protocol used by them to compute $EQ(a,b) = \mathbb{I}[a = b]$ must exchange at least N bits.*

Fact 3. *[INDEX [78]] If Alice and Bob have two inputs $a \in \{0,1\}^N$ and $b \in [N]$ respectively, then any deterministic communication protocol used by Alice to send bits to Bob must require N bits for Bob to compute $\textsc{Index}(a,b) = a_b$. Moreover, the one-way randomised communication complexity of the INDEX is $\Omega(N)$.*

One thing to note about the Equality problem is that although the deterministic communication complexity of the EQ problem is $\Omega(N)$, the randomised communication complexity is $O(\log N)$. For the Disjointness and INDEX problems, the randomised communication complexity is $\Omega(N)$ as well. In other words, even if the two parties are allowed to compute the output of the function correctly with high probability (say $> 2/3$), even then the number of bits they must exchange is $\Omega(N)$.

3.2.2 Finite Precision Implementation

In this work, we are interested in the expressiveness of finite precision models. For our constructions with Transformers, we will work with p-bit numbers where $p = \Theta(\log N)$ where N is the maximum length of the input string.
In particular, for some sufficiently large constant $\mathrm{K}_c > 0$, we will work with numbers between $-N^{\mathrm{K}_c}$ to N^{K_c} with a step size of $\Delta = \frac{1}{N^{\mathrm{K}_c}}$. If $\mathbb{Q}_p$ is the set of all such numbers then $\mathbb{Q}_p = \{-N^{\mathrm{K}_c}, -N^{\mathrm{K}_c} + \Delta, \ldots, 0, \Delta, 2\Delta, \ldots, N^{\mathrm{K}_c}\}$. Hence, the size of the set is $|\mathbb{Q}_p| = N^{2\mathrm{K}_c} \implies p = 2\mathrm{K}_c \log N = \Theta(\log N)$. For any real number $z \in \mathbb{R}$, in the finite precision implementation, z is rounded down to the nearest $\hat{z}$ such that $\hat{z}N^{\mathrm{K}_c} \in \mathbb{Z}$. If $z > N^{\mathrm{K}_c}$, then z is rounded down to N^{K_c}. In our constructions with Transformers, all the parameters and intermediate values follow the above implementation.

3.2.3 Technical Tools

For our constructions of Transformers, we will use the following result about high-dimensional vectors. The statement essentially says that at a high dimension k the number of vectors that are almost orthogonal to each other is exponential in k even though the number of orthogonal vectors can be at most k.

Lemma 1. *For any N, there exists N k-dimensional vectors $\mathcal{T}_1, \ldots, \mathcal{T}_N$ where $k = O(\frac{1}{\gamma^2} \log N)$ and each entry of the vectors is in $\{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}$ such that*

$$\langle \mathcal{T}_i, \mathcal{T}_j \rangle \begin{cases} \geq 1 - \gamma & \text{if } i = j, \\ \leq \gamma & \text{otherwise.} \end{cases}$$

It is quite straightforward to see why this is true. It follows from a simple application of the probabilistic method. Suppose one samples N k-dimensional vectors $\mathbf{X}_1, \ldots, \mathbf{X}_N$ independently at random such that each entry of each vector is drawn uniformly from $\{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}$. Hence, each vector $\mathbf{X}_i \in \{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}^k$ and the expectation of the dot product of any two vectors $\mathbb{E}[\langle \mathbf{X}_i, \mathbf{X}_j \rangle] = 0$ for all $i \neq j$. Since the dot products of any two vectors $\langle \mathbf{X}_i, \mathbf{X}_j \rangle$ is a bounded random variable, we can apply Hoeffding's inequality to get

$$\mathbb{P}[|\langle \mathbf{X}_i, \mathbf{X}_j \rangle| \geq \gamma] \leq 2 \exp\left(\frac{-k\gamma^2}{4}\right).$$

Taking a union bound over at most N^2 pairs of dot products and setting $2N^2 \exp\left(\frac{-k\gamma^2}{4}\right) < 1/2$, we get that for $k = \frac{8}{\gamma^2} \log 2N$, the probability of all dot products $\langle \mathbf{X}_i, \mathbf{X}_j \rangle$ being

less than γ at the same time is at least $1/2$. Since the probability of the event is nonzero, it follows that the statement in the lemma is true.

In our construction, we will use such vectors as positional embedding vectors. While Lemma 1 implies the existence of vectors, storing N such vectors will require $\Theta(N)$ space. We would like to generate i-th vector $\mathcal{T}_i$ when necessary in polynomial time using log space without storing all of them together. To achieve that, we use a derandomisation of the Johnson-Lindenstrauss (JL) Lemma [80] that can output the i-th vector in log-space and polynomial time [144].

We use a slightly modified version of the JL lemma which preserves inner products over unit vectors.

Lemma 2. *[Inner product preservation [1]] Let $\epsilon \in (0, 1/2)$ and let $\mathcal{Q} \subset S^{d-1}$ be a set of N unit norm vectors of dimension d. For $k = \frac{c_2 \log N}{\epsilon^2}$, there exists a linear map $\mathcal{T}(x) = \frac{1}{\sqrt{k}}Ax$ where each entry of $A : \mathbb{R}^d \to \mathbb{R}^k$ is in $\{-1, 1\}$ such that for all $u, v \in \mathcal{Q}$,*

$$|\langle u, v\rangle - \langle \mathcal{T}(u), \mathcal{T}(v)\rangle| \leq \epsilon$$

The above result has interesting implications which we will use in our constructions. Let $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^N$ be N unit norm vectors that form the basis of $\mathbb{R}^N$ which implies $\langle \mathbf{x}_i, \mathbf{x}_j\rangle = 1$ if $i = j$ and is 0 otherwise. Then there exists a map $\mathcal{T}$ such that the vectors $\mathcal{T}(\mathbf{x}_1), \ldots, \mathcal{T}(\mathbf{x}_N) \in \mathbb{R}^k$ have dot products $\langle \mathcal{T}(\mathbf{x}_i), \mathcal{T}(\mathbf{x}_j)\rangle = 1 \pm \epsilon$ if $i = j$ and is $0 \pm \epsilon$ otherwise.

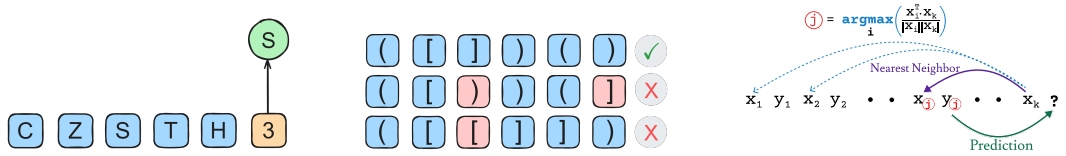
We will use JL transformations of the standard basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_N$ where $\mathcal{T}(1), \ldots, \mathcal{T}(N)$ will refer to $k = O(\log N)$ dimensional vectors such that their inner product is ≈ 1 with themselves and is ≈ 0. Intuitively, we can use such vectors to get a Transformer to attend to unique positions.

Corollary 0.1. *For any N, there exists N k-dimensional vectors $\mathcal{T}(1), \ldots, \mathcal{T}(N)$ where $k = O(\log N)$ and each entry of the vectors is in $\{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}$ such that*

$$\langle \mathcal{T}(i), \mathcal{T}(j)\rangle \begin{cases} \geq 3/4 & \text{if } i = j, \\ \leq 1/4 & \text{otherwise.} \end{cases}$$

3.3 Index Lookup Task

Task Description. We introduce a simple task called Index Lookup (IdxL). In this task, a model receives a sequence of tokens $s_1, \ldots, s_N$ (possibly with repetitions)



(a) Index Lookup Task (b) Dyck-2 with depth $\leq$ 2 (c) Nearest Neighbour Task

Figure 3.1: Illustration of a few key tasks considered in our work.

followed by an index p where $p \in [N]$ and the goal of the model is to output the token s_p. Here, the symbols s_i belong to a finite alphabet Σ.

This simple and natural task helps illustrate the key tools and building blocks we use to obtain other more general results in this chapter: On the one hand, we show how a one-layer Transformer with width $O(\log N)$ can perform this task; on the other hand, we use communication complexity arguments to show that any type of recurrent or state-space model performing this task needs a hidden state with representation size $\Omega(N)$.

Our first result shows that, for any length $N \in \mathbb{N}$, there is a 1-layer Transformer with width $O(\log N)$ that performs the Index Lookup task for all input sequences of length at most N. Naïvely one could construct such a transformer by using one-hot encodings as positional embeddings, as they are orthogonal and would allow to attend to the desired index. However, this would require the embedding dimension, and hence the width of the model, to be $\Omega(N)$. Key to our constructions of a width $O(\log N)$ Transformer, both here and in other sections, is Lemma 1, which states that, in $k = O(\log N/\gamma^2)$ dimensional space we can find N nearly orthogonal vectors. We use such vectors in our construction of the Transformers to allow it to attend almost exactly over the desired positions.

Theorem 1. *For all $N \in \mathbb{N}$, there is a 1-layer Transformer with width $m = O(\log N)$ and precision $p = O(\log N)$ which performs the index lookup task for all input sequences of lengths up to N.*

Proof. For an input sequence $(s_1, \ldots, s_N, p)$, the Transformer uses the embeddings of the position token p and the positional embeddings of the first N inputs to attend over s_p, so that the feedforward network can extract the label from the output of the attention block. Our key idea is to use the N almost orthogonal vectors provided by Lemma 1, both as positional embeddings and also as a way to embed the numbers $\{1, \ldots, N\}$, any of which can be used as the index p. Formally, let $\mathcal{T}(1), \ldots, \mathcal{T}(N)$

be N vectors of dimension $k = O(\log N)$ such that $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \leq 1/4$ for $i \neq j$ and $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \geq 3/4$ for $i = j$.

Recall that, in the index lookup task, the alphabet $V = \Sigma \cup [N]$ consists of symbols s_i from a set Σ and the index tokens $[N] = \{1, \ldots, N\}$. The embeddings of each input token will be of size $\log |\Sigma| + 2k$ where $\log |\Sigma| + k$ indices are reserved for the token embeddings and the last k indices are used for the positional embeddings. Suppose we use a binary encoding for symbols in Σ, and $\rho : \Sigma \to \{0,1\}^{\log |\Sigma|}$ is such that $\rho(s)$ is the binary encoding of $s \in \Sigma$. The token embedding of each symbol $s_j \in \Sigma$ is $\rho(s_j)$ of length $\log |\Sigma|$ followed by k zeros. For input sequence $s_1, \ldots, s_N$, the embedding vectors will be of the form $\mathbf{x}_i = [\rho(s_i), \mathbf{0}_k, \mathcal{T}(i)]$.

The embedding for any index token $p \in [N]$ contains $\log |\Sigma|$ zeros followed by the vector $\mathcal{T}(p)$. In other words, the embedding for the index token will be identical to the positional embeddings used in the first N tokens. The embedding vector corresponding to the index token p will be of the form $\mathbf{p} = [\mathbf{0}_{\log |\Sigma|}, \mathcal{T}(p), \mathbf{0}_k]$.

The output of the 1-layer Transformer is computed by applying attention over the input sequence with the query vector corresponding to the last token, followed by an application of the feedforward network over the output of the attention block. We can define the matrices $W_Q = \eta[\mathbf{O}; \mathbf{I}; \mathbf{O}]$, $W_K = [\mathbf{O}, \mathbf{O}, \mathbf{I}]$, $W_V = [\mathbf{I}, \mathbf{O}, \mathbf{O}]$, where $\eta > 0$ is a parameter that will be specified later, $\mathbf{I}$ is a square identity matrix and $\mathbf{O}$ is a zero-matrix of the appropriate shape. With these definitions we get that the query vector $Q(\mathbf{p}) = \eta[\mathcal{T}(p)]$ contains the middle part of the input embedding, the key vectors $K(\mathbf{x}_i) = [\mathcal{T}(i)]$ contain the last part of the input embedding and the value vectors $V(\mathbf{x}_i) = [\rho(s_i)]$ contain the first part of the input embedding.

With these query and key vectors, the dot products in attention satisfy:

$$\langle Q(\mathbf{p}), K(\mathbf{x}_i) \rangle \begin{cases} \geq 3\eta/4 & \text{if } i = p, \\ \leq \eta/4 & \text{if } i \neq p \end{cases}.$$

Additionally, the dot product of the query vector with itself will be $\langle Q(\mathbf{p}), K(\mathbf{p}) \rangle = 0$.

To retrieve the required token with the softmax operator, consider

$$\begin{aligned} \text{softmax}(Q(\mathbf{p}), K(X)^\top)_p =& \frac{\exp(\langle Q(\mathbf{p}), K(\mathbf{x}_p) \rangle)}{\exp(\langle Q(\mathbf{p}), K(\mathbf{x}_p) \rangle) + \sum_{j \neq p} \exp(\langle Q(\mathbf{p}), K(\mathbf{x}_j) \rangle)} \\ \geq& \frac{\exp(\frac{3\eta}{4})}{\exp(\frac{3\eta}{4}) + N \exp(\frac{\eta}{4})} \end{aligned}$$

which is $> \frac{3}{4}$ for any $\eta > 2 \log(3N)$. That is, for $\eta > 2 \log(3N)$, the attention weight over input token x_p with a query token p will be greater than $3/4$, and hence the total attention weight over the remaining tokens will be less than $1/4$.

Recall that the value vectors contain the binary encodings of the input symbols, $V(\mathbf{x}_i) = [\rho(s_i)]$. Let $q_1, \ldots, q_N$ be the probabilities assigned to the N tokens by the attention operator. Note that $q_p \geq 3/4$. Let $\bar{z} = \sum_i q_i \rho(s_i)$. Note that if the j-th bit of $\rho(s_p)$ is 1, then $\bar{z}_j \geq 3/4$ and otherwise, $\bar{z}_j \leq 1/4$. From there a ReLU-FFN can transform the vector $\bar{z}$ to the vector $\rho(s_p)$ which is the desired output. □

We use results from communication complexity to show that RNNs require essentially $\Omega(N)$ width to solve this and several other problems. In communication complexity, there are two parties, typically called Alice and Bob, each of whom has part of the (discrete) input, and their goal is to compute a function of the combined input using as little communication as possible. Our first key insight here is that the output of an RNN can be computed with a bounded amount of communication when Alice has a prefix of the input and Bob has the remaining part. The resulting protocol will be one-way (Alice to Bob) and one-round. We first state a more general result and then discuss implications for the IdxL problem.

Theorem 2. *If an RNN with a hidden state of representation size mp computes any function $f : \Sigma^N \to \{0, 1\}$, then for any $K < N$, if Alice has access to $s_1, \ldots, s_K$ and Bob has access to $s_{K+1}, \ldots, s_N$, then there exists a one-way communication protocol with mp bits from Alice to Bob, by which Bob can compute the output of the function $f(s_1 \ldots s_N)$.*

Proof. Assume that both Alice and Bob have access to the RNN that represents the function f. Alice can provide the sequence $s_1, \ldots, s_K$ to the recurrent model and iteratively update the hidden state from the initial state $\mathbf{h}_0$ to obtain the Kth hidden state $\mathbf{h}_K$. Alice can then send the hidden state to Bob which requires mp bits. Bob can then update the hidden state using $s_{K+1}, \ldots, s_N$ to obtain $\mathbf{h}_N$, from which he can obtain the output of the RNN. Note that Alice and Bob can compute the output using one-way communication of mp bits. □

Problems similar to Index Lookup are well-studied in communication complexity; specifically, the INDEX problem (See Section 3.2.1) has a one-way communication complexity of $\Omega(N)$ (Fact 3). We deduce a lower bound on the size of the hidden state of RNNs by showing that any RNN that can represent the Index Lookup task can also compute the INDEX problem and since that implies the existence of a one-way communication protocol with mp bits (Theorem 2), it follows that the width of the hidden state m must be $\Omega(N/p)$.

Theorem 3. *Any recurrent model with a hidden state of width m using p bits of precision that computes the Index Lookup task for all sequences of length N must have $m \geq N/p$.*

Proof. The proof follows naturally via a reduction from the INDEX problem in communication complexity. Assume the vocabulary size for the IdxL is at least 2, pick any two symbols from the vocabulary, and map them to 0 and 1. Suppose Alice has a sequence $a \in \{0,1\}^N$ and Bob has an index $i \in [N]$. Both of them have access to a recurrent model as described in Section 3.2, which can perform the IdxL task perfectly. Alice can then provide the sequence a to the recurrent model using any two symbols in the vocabulary Σ and iteratively update the hidden state to obtain the Nth hidden state h_N. Alice can then send the hidden state to Bob, which requires mp bits. Bob can then provide the position token based on the index i and compute the output to figure out whether a_i is 0 or 1.
Note that Alice and Bob can compute the output using one-way communication of mp bits, and hence, based on Fact 3, it must be the case that $mp \geq N$.

□

Discussion. The above results theoretically formalise intuitive differences between the way Transformers and recurrent models process sequences. Since Transformers have access to N input vectors during their computation, a small-sized attention block can attend over the desired input vector to make the correct prediction. On the other hand, any recurrent model—even with arbitrary positional embeddings—must store all the required information in its hidden state, which lower bounds the size of such models to compute the right output. These intuitions are made rigorous by showing (i) how soft-attention can do lookup using the almost orthogonal vectors, and (ii) small-width RNNs yield a short one-way communication protocol. These lower bounds also apply to causal forms of linear attention architectures where softmax is removed and attention weights become dot products [82].
At first glance, it might seem unfair to compare Transformers and RNNs with the same number of parameters: Transformers have access to N input vectors, whereas RNNs have a fixed-size hidden state. But note that, in practice, empirical research on language models typically compares models of the same size e.g., a 7B Transformer vs a 7B state-space model. Hence, it is natural to ask if Transformers of a particular size can express something that recurrent models cannot.

3.4 Lower Bounds for RNNs and 1-layer Transformers

Whereas Section 3.3 established a case where one-layer Transformers can be more powerful than RNNs, we next exhibit an example of the opposite phenomenon. Here, the key tool will again be a communication complexity argument, but this time it applies to one-layer Transformers: We establish a communication protocol by which Alice and Bob can compute the output of a one-layer Transformer by exchanging a number of bits that is bounded by the representation size of the Transformer and an overhead that is logarithmic in the input length. The key property here is that this protocol works not just when Alice and Bob have access to a prefix and a suffix of a string, but instead works for an *arbitrary* partitioning of the input string.
As described in Section 3.2.2, and in keeping with real-world implementations, the outputs of intermediate computations are rounded to p-bit precision. Further in keeping with real implementations, and to avoid overflow in exponentiation, softmax is implemented by first subtracting the maximum logit, as

$$\mathrm{softmax}(A)_{i,j} = \frac{\exp(A_{i,j} - \max_l A_{i,l})}{\sum_{k=1}^{M} \exp(A_{i,k} - \max_l A_{i,l})}.$$

This is a popular approach to implementing softmax [25]; it ensures that the exponentiated intermediate results are in $[0, 1]$, avoiding possible overflow when exponentiating in finite precision.

Theorem 4. *Consider a one-layer Transformer $f \in TF^1_{m,p,H}$ operating over inputs of length N. Consider any disjoint subsets $S_A \cup S_B = \{1, \ldots, N\}$, $S_A \cap S_B = \emptyset$. Assume Alice has access to s_i for $i \in S_A$, and Bob has access to s_i for $i \in S_B$. Then Alice and Bob can communicate $3m(p + \log N)H$ bits to compute the output $f(s_1 \ldots s_N)$.*

We note that conceptually related arguments were used in Sanford et al. [134, Theorem 7] and Peng et al. [123, proof of Theorem 1]). Our approach here generalises by stating this for *arbitrary* partitions over the input.

Proof. Without loss of generality, assume that $N \in S_A$, i.e., Alice has access to x_N. In the *first step*, Alice sends x_N to Bob using dp bits of communication. Then, Alice and Bob compute for each head the attention logits for each position within their respective sets:

$$A_{N,i} := \langle Q(\mathbf{x}_N), K(\mathbf{x}_i) \rangle \tag{3.1}$$

These numbers are rounded to p bits of precision.

In the *second step* of the protocol, Alice and Bob exchange $2p$ bits to determine $M := \max_i A_{N,i}$, and compute

$$\widehat{A}_{N,i} = A_{N,i} - M \tag{3.2}$$

for their respective positions $i \in S_A, S_B$. Note that, as $\widehat{A}_{N,i} \leq 0$, $\exp(\widehat{A}_{N,i}) \in (0, 1]$, so there are no overflow issues arising from the p-bit representation. Then they can individually compute

$$Z_A := \sum_{i \in S_A} \exp(\widehat{A}_{N,i}) \qquad \text{(Alice)}$$
$$Z_B := \sum_{i \in S_B} \exp(\widehat{A}_{N,i}) \qquad \text{(Bob)}$$

each with p bits of precision; both numbers are in $[0, N]$, and at least one of them is ≥ 1. As all intermediate computations are rounded to p bits, $\exp(\widehat{A}_{N,i})$ is in $[0, 1]$ and rounded to p bits, Z_A, Z_B can be represented with $p + \log N$ bits.

In the *third step* of the protocol, they exchange $2(p + \log N)$ bits to exchange these. They then both have access to

$$Z := \sum_{j=1}^{N} \exp(\widehat{A}_{N,j}) = Z_A + Z_B \tag{3.3}$$

Here, we note that the definition of finite precision arithmetic in Section 3.2.2 makes addition of nonnegative numbers associative; hence, the outcome here is independent of the partition S_A, S_B and agrees with the result when directly summing the exponentiated logits.[1]

The result Z is in $[1, N]$ as $\max_i \exp(\widehat{A}_{N,i}) = 1$. This permits Alice and Bob to compute the attention scores $\widehat{a}_i$:

$$\widehat{a}_i := \text{softmax}(A)_{N,i} = \frac{\exp(\widehat{A}_{Ni})}{\sum_{j=1}^{N} \exp(\widehat{A}_{Nj})} \tag{3.4}$$

each with p bits of precision.

Then they each compute:

$$U_A := \sum_{i \in S_A} \text{softmax}(A)_{N,i} V(\mathbf{x}_i) \qquad \text{(Alice)}$$
$$U_B := \sum_{i \in S_B} \text{softmax}(A)_{N,i} V(\mathbf{x}_i) \qquad \text{(Bob)}$$

[1] A similar protocol still works in other bounded-precision schemes where addition is not associative; as all exponentiated logits are in $[0, 1]$, one can use extended precision with $p + \log N$ bits to perform the summation exactly and then round back to p bits.

with p bits of precision. As each entry $\mathrm{Att}(A)_{N,i}$, $V(\mathbf{x}_i)$ has p bits of precision, and $\sum_i \mathrm{Att}(A)_{N,i} = 1$, the sums can be exactly represented at $2p$ bits of precision.
In the *fourth step* of the protocol, they exchange these, which amounts to $4mp$ bits of communication. Then they compute

$$\sum_{i=1}^{N} \mathrm{softmax}(A)_{N,i} V(x_i) = U_A + U_B \tag{3.5}$$

and round it to p bits of precision. From this, they can obtain the result.
In total, the four steps took $\leq 3m(p + \log N)$ bits of communication. Performing this protocol separately for every head leads to $\leq 3m(p + \log N)H$ bits of communication.
□

3.4.1 Separation on Bounded Hierarchical Languages

We now use the communication protocol for one-layer Transformers to establish a separation between these and RNNs on bounded Dyck languages. Dyck languages are of central importance in formal language theory, since the Chomsky–Schützenberger theorem [40] shows that every context-free language can be written as the homomorphic image of the intersection of a Dyck language with a regular language. Thus Dyck languages can be viewed as canonical building blocks for context-free structure. This is one of the primary motivations for using Dyck languages as a testbed in this work, as well as in prior works [152, 19, 72]. Due to the boundedness of human memory [104], natural language tends to have more bounded levels of embedding [81, 26]. This has motivated the study of bounded-depth Dyck languages as plausible simple models of the hierarchical structure underlying language [72, 175, 19, 170].
Task. Formally, Dyck-(n, k) is the language of well-matched strings over n types of parenthesis pairs $(_1,)_1, (_2,)_2, \ldots, (_n,)_n$, where any prefix has at most k opening parentheses not yet closed. For instance, the string '([ ]) ( )' has a maximum depth 2 corresponding to the prefix '(['. Dyck-(n, k) can be recognised with access to a bounded stack that never holds more than k elements. In fact, each Dyck-(n, k) is a regular language and is accepted by a finite automaton.
We show that there is a linear communication complexity lower bound for Dyck-(n, k), already at $n = k = 2$. However, unlike the communication bound we used in Theorem 3, Alice and Bob now have access not to two halves of the input; rather, Alice and Bob have access to the even and odd positions in the string, respectively. Intuitively, in such a situation, Alice needs to know almost all of the bits available to Bob in order

to decide whether a given string is well-bracketed–and vice versa. More formally, they need to exchange at least $N-1$ bits to decide membership in DYCK-(2, 2).

Problem. Let $\Sigma = \{`(', `)', `[', `]'\}$ be the vocabulary of a language DYCK-(2, K). Let Σ^n denote the set of all strings of length exactly n and $\Sigma^{\leq n}$ denote the set of all strings of lengths up to n. The communication problem between Alice and Bob is defined as follows. Assume the length N is even for this problem. For a string $x \in \Sigma^N$, Alice has the symbols in the odd indices of the string and Bob has the symbols in the even indices of the string. They have to compute the function $f_{\mathsf{Dyck}} : \Sigma^{N/2} \times \Sigma^{N/2} \to \{0,1\}$ which outputs 1 if the string x is in the language DYCK-(2, 2) and outputs 0 otherwise.

For any function, the fooling set is defined in the following way,

Definition 1. [Fooling set] A fooling set for any function $f : \Sigma^{N/2} \times \Sigma^{N/2} \to \{0,1\}$ is a set $S \subseteq \Sigma^{N/2} \times \Sigma^{N/2}$ and a value $b \in \{0,1\}$ such that,

- For every $(x,y) \in S$, $f(x,y) = b$.
- For every two distinct pairs $(x_1,y_1),(x_2,y_2) \in S$, either $f(x_1,y_2) \neq b$ or $f(x_2,y_1) \neq b$.

The following is a well-known fact in communication complexity.

Fact 4. *For any function f, if there exists a fooling set of size $|S|$, then the communication complexity $C(f) \geq \log_2 |S|$.*

Lemma 3. *Suppose Alice and Bob have the symbols in the odd and even indices of a string $s \in \Sigma^N$ respectively. To each compute whether $s \in$ DYCK-(2, 2), they must exchange at least $N-1$ bits.*

Proof. The proof follows from the fact that there exists a fooling set of size 2^{N-1} for the language DYCK-(2, 2) with strings of length N. The fooling set S is constructed with all strings in $s = (x,y) \in \Sigma^N$ such that $f_{\mathsf{Dyck}}(s) = 1$. Each string s in S satisfies:

$$s = (x,y) \quad \text{where} \quad x \in \Sigma^{N/2}_{\text{odd}}, y \in \Sigma^{N/2}_{\text{even}}, \text{ and } f_{\mathsf{Dyck}}(x,y) = 1.$$

Here, $x = (x_1, x_2, \ldots, x_{N/2})$ and $y = (y_1, y_2, \ldots, y_{N/2})$ represent the sequences of symbols at odd and even indices, respectively.

Constructing the fooling set. Note that, if x is the string of symbols in the odd indices of a string $s \in$ DYCK-(2, 2), then the string of symbols y in the even indices such that $f_{\mathsf{Dyck}}(x,y) = 1$ is unique. Suppose one is provided with the string $x = (x_1, \ldots, x_{N/2})$, then one can deterministically determine the symbols in the even

indices $y = (y_1, \ldots, y_{N/2})$ in the following way. Iterate through the symbols in x, starting with x_1, which must be an open bracket. After the open bracket x_1, if there are one or more closing brackets $x_2, \ldots, x_K$ before encountering another open bracket x_{K+1}, then the symbols $y_1, \ldots, y_K$ can be constructed deterministically using the following mapping,

$$y_j = \begin{cases} \text{`('} & \text{if } x_{j+1} = \text{`)' for } j < K, \\ \text{`['} & \text{if } x_{j+1} = \text{`]' for } j < K, \\ \text{`)'} & \text{if } x_1 = \text{`(' for } j = K, \\ \text{`]'} & \text{if } x_1 = \text{`[' for } j = K. \end{cases}$$

In other words, the symbols $y_1, \ldots, y_{K-1}$ will be the open brackets corresponding to the closing brackets $x_2, \ldots, x_K$. After the symbol x_1, whenever you encounter another open bracket x_{K+1} where $K > 0$, then the symbol y_K will be the closing bracket corresponding to the symbol x_1. Once you have matched the first open bracket symbol x_1 with a closing bracket, you are bound to encounter another open bracket. Follow the same process until the end of the string, and one can obtain the string y, such that $(x, y) \in$ DYCK-(2, 2).

Hence, if x and y are the symbols in the odd and even indices of a string $s \in$ DYCK-(2, 2), then placing any other string $y' \neq y$ in the even indices leads to a string which is not in DYCK-(2, 2). Our fooling set S contains all strings of length N in the language DYCK-(2, 2). Hence, by construction, we have that $f_{\mathsf{Dyck}}(s) = f_{\mathsf{Dyck}}(x, y) = 1$ for all $s = (x, y) \in S$ and

$$f_{\mathsf{Dyck}}(x_1, y_2) = f_{\mathsf{Dyck}}(x_2, y_1) = 0 \quad \text{for all} \quad (x_1, y_1) \neq (x_2, y_2) \in S.$$

Thus, such a set S is a fooling set by Definition 1.

Size of the fooling set. One can show that the total number of strings of length N in the language DYCK-(2, 2) is exactly 2^{N-1} with some elementary combinatorics. First, see that the number of DYCK-2 sequences of depth at most 1 and length N is exactly $2^{N/2}$. This is because the string is made up of blocks of '()' and '[]', and hence there are two choices for every block. Then, consider a block of DYCK-2 string of length k (where k is even) which starts and ends with an open and closing bracket, respectively. Moreover, the DYCK-2 string has a depth exactly 2, e.g. '[() [] ()]'. Note that, the total number of such strings of length k is exactly $2^{k/2}$ since there are two choices for the first bracket and there is a DYCK-2 string of depth 1 and length $k - 2$ inside it. For simplicity, we will call such strings as belonging to the language DYCKB-(2, 2) which is a subset of the language DYCK-(2, 2). In simple

terms, strings of length m in the subset DYCKB-(2, 2) have an overall depth 2 and have a depth 1 string of length $m-2$ between the first symbol (open bracket) and the last symbol (closing bracket); for instance, '[() [] () ()]'.

The total number of DYCK-2 strings of depth at most 2 and length exactly N can be computed as follows. Partition the indices into k contiguous blocks where each partition is of an even length. Suppose the indices begin with 1, then the points of partition could be between 2 and 3, 4 and 5, and so on. For instance, a valid partition of the indices $[1, 2, \ldots, 8]$ into 3 blocks is $[1, 2]$, $[3, \ldots, 6]$, and $[7, 8]$. The total number of partition points is $N/2 - 1$. For any such partition, if the length of a block is 2 then that block can only have DYCK-2 strings of depth 1 and if the block is of length ≥ 4, then consider all possibilities of DYCK-(2, 2) strings starting and ending with open and closing brackets respectively or in other words, strings in DYCKB-(2, 2) described earlier. If the number of partitions is $N/2 - 1$, then all possible strings have depth at most 1 and if the number of partitions is 0, then the entire string is in DYCKB-(2, 2). See that, no matter how you partition the inputs, the number of possible strings at each block of length m is $2^{m/2}$. Further if $P = \{p_1, \ldots, p_k\}$ denotes the set of lengths of each block in the partition, then the total number of strings with such a partition is $\prod_{i=1}^{k} 2^{p_i/2} = 2^{\sum_{i=1}^{k} p_i/2} = 2^{N/2}$. In other words, regardless of how the indices are partitioned, if each block of size > 2 is required to be a string in DYCKB-(2, 2), then the total number of possible strings is $2^{N/2}$.

The total number of DYCK-2 strings of depth at most 2 can be computed by considering partitions of all sizes $k = 0, \ldots, N/2 - 1$ and all possible partitions for a given size. With this, we get the total number of valid strings in DYCK-(2, 2) of length N to be

$$\sum_{k=0}^{\frac{N}{2}-1} \binom{\frac{N}{2}-1}{k} 2^{N/2} = 2^{N-1}$$

Hence, from fact 4, it follows that the communication complexity of DYCK-(2, 2) is at least $N - 1$.

□

While we have given a self-contained proof based on fooling sets, there is also a possible algebraic route to Lemma 3. The result of Raymond et al. [131, Theorem 5.7] says that for languages whose syntactic monoid is aperiodic, the two-party communication complexity is $\Theta(\log N)$ when the monoid lies in DA, and $\Theta(N)$ when it is not in DA. Since DYCK-(2, 2) is star-free, this would reduce the question to showing that its syntactic monoid is not in DA. Informally, this looks plausible because DYCK-(2, 2)

is defined by nested bracket-matching constraints, rather than by a simple left-to-right pattern. Thus establishing that the syntactic monoid of bounded Dycks is not in DA, the linear lower bound would follow from the result of Raymond et al. [131].
Combining Theorem 4 and Lemma 3 entails a lower bound on the width of a Transformer for computing DYCK-(2, 2):

Theorem 5. *Consider a one-layer Transformer $f \in TF^1_{m,p,H}$ deciding membership in DYCK-(2, 2). Then $mH \geq \frac{N-1}{3(p+\log N)}$.*

Discussion. This result establishes a second separation between one-layer Transformers and RNNs, but now in the other direction: Bounded-depth Dyck languages are regular, and previous work has shown that constant-width RNNs can recognise them with standard activation functions, Sigmoid [72] and ReLU [19]. We further note that two-layer Transformers can model bounded-depth Dyck languages [175]. It is also worth noting that the argument for the communication complexity lower bound for Dyck with 2 types of brackets can also be naturally adapted to Dyck with 1 type of brackets. Thus, it also implies that one-layer Transformers need linear width to represent Dyck-1 and a two-layer construction (such as one based on Bhattamishra et al. [18]) is needed.

3.4.2 Lower Bounds on Boolean Functions

There are some notable differences between the types of communication complexity lower bounds we have for one-layer Transformers (Theorem 4) and for RNNs (Theorem 2). If an RNN can compute a function f then Theorem 2 implies that there exists a one-way communication protocol with mp bits over any *contiguous* partitions of the input. On the other hand, if a one-layer Transformer can compute a function f, then Theorem 4 implies the existence of a communication protocol with $O(mpH \log N)$ bits over *any partition* of the input — but not one-way. Hence, the types of lower bounds that apply to these two architectures are different. For any function f such as the INDEX problem, if the one-way communication complexity is lower bounded by $\Omega(N)$ over some contiguous partitions, then it implies that for RNNs, the width $m = \Omega(N/p)$. However, since the two-way communication complexity of such problems might be smaller, those lower bounds need not apply to one-layer Transformers. For instance, the INDEX problem can be solved with $\log N$ bits of communication if two-way communication is allowed or in other words, Bob is allowed to send bits to Alice. Conversely, to prove lower bounds for Transformers computing a certain function f, proving a communication complexity lower bound for f over any partition

of inputs suffices. For a particular partitioning of inputs, we showed a lower bound on the communication complexity of bounded Dyck languages. While that result implies a lower bound on one-layer Transformers, the partitioning is not contiguous and hence does not apply to recurrent models. For DYCK-(2, 2), contiguous partitions are not a hard case, and in fact, communicating ≤ 2 open brackets from the first half is sufficient. This is why the lower bound of Lemma 3 does not apply to RNNs. Despite these differences, for a large class of Boolean functions including functions such as Equality and Disjointness, the lower bounds apply to both of these architectures.

Lower Bounds. Despite the differences discussed above, there are several Boolean functions, for which we can establish that when widths are bounded, neither one-layer Transformers nor RNNs can compute them. The Equality function $\text{EQ} : \{0,1\}^N \to \{0,1\}$ is a Boolean function defined as $\text{EQ}(\mathbf{x}) = \mathbb{I}[(\mathbf{x}_1, \ldots, \mathbf{x}_{N/2}) = (\mathbf{x}_{N/2+1}, \ldots, \mathbf{x}_N)]$. A related problem is *Disjointness*: given two vectors $\mathbf{x}, \mathbf{y} \in \{0,1\}^{N/2}$, the function $\text{DISJ}(\mathbf{x}, \mathbf{y}) = \max \mathbf{x}_i \mathbf{y}_i = \mathbb{I}[\mathbf{x}^T \mathbf{y} > 0]$. Both the functions Equality and Disjointness are known to have communication complexity $\Omega(N)$ (see Section 3.2.1) and Theorems 4 and 2 imply that both one-layer Transformers and RNNs must have width $\Omega(N)$ to represent them. In the next section, we show that these lower bounds do not apply to two-layer Transformers. Additionally, it is worth noting that the functions EQ and DISJ can also be expressed as depth-2 linear-size circuit families, namely as a 2-CNF and a 2-DNF respectively. Hence, a more general consequence of the limitations of RNNs and one-layer Transformers is that with width $o(N)$, they cannot compute certain functions in the class DLOGTIME-uniform AC^0.[2] This is interesting since DLOGTIME-uniform AC^0 is considered one of the simplest classes of Boolean circuits.

3.5 Representational Capabilities of 2-layer Transformers

In Section 3.4, we showed that single-layer Transformers and recurrent models must have size *linear* in the input length to express natural Boolean functions such as EQ. In this section, we show that two-layer transformers overcome these limitations by efficiently expressing such Boolean functions and more general forms of associative recall tasks, such as simulating the nearest neighbour algorithm.

[2]The class AC^0 contains polynomial size AND/OR circuits with unbounded fan-in and constant depth.

3.5.1 Representing Boolean Functions

We start by showing that two-layer Transformers of poly-logarithmic size can express the Equality function. The input domain need not necessarily be the Boolean vectors $\{0,1\}^N$; rather, the construction works for sequences over any finite alphabet Σ.

Equality. For convenience, we discuss the complement of the Equality function $\text{INEQ} = 1 - \text{EQ}(x)$, which is the inequality function. It does not influence any of our results or the constructions for Transformers but it helps make the intuitions for the construction clearer. The function $\text{INEQ} : \{0,1\}^N \to \{0,1\}$ is a Boolean function defined as,

$$\text{INEQ}(x) = \mathbb{I}[(x_1, \ldots, x_{N/2}) \neq (x_{N/2+1}, \ldots x_N)]$$

This can also be represented as a 2-DNF with $N/2$ terms,

$$\text{INEQ}(x) = (x_1 \wedge \neg x_{N/2+1}) \vee (\neg x_1 \wedge x_{N/2+1}) \vee \ldots \vee (x_{N/2} \wedge \neg x_N) \vee (\neg x_{N/2} \wedge x_N)$$

Theorem 6. *For any $N \in \mathbb{N}$, there exists a 2-layer Transformer $f \in TF^2_{m,p,2}$ where width $m = O(\log N)$ and precision $p = O(\log N)$ such that $f(\mathbf{x}) = \text{INEQ}(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^N$.*

The construction is based on tools developed in Section 3.3. The broad idea is as follows. In the first layer, at each position $i > N/2$, an attention head attends to position $i - N/2$ and copies the input $x_{i-N/2}$. A feedforward network then checks whether the retrieved value is equal to x_i. The second layer simply uses uniform attention over the outputs of the previous layer to check if there is a mismatch at any position. Importantly, we show that the above strategy can be implemented with a representation size $O((\log N)^3)$.

We first describe the high-level idea behind the construction. We will consider the input domain to be $\{-1,1\}^N$ and our goal is to compute $\text{INEQ}(x)$ for all $x \in \{-1,1\}^N$. This can also be formulated as,

$$\text{INEQ}(x) = (x_1 \oplus x_{\frac{N}{2}+1}) \vee (x_2 \oplus x_{\frac{N}{2}+2}) \vee \ldots \vee (x_{\frac{N}{2}} \oplus x_N)$$

Let $f^{(1)}_{\text{ineq}}$ denote the first layer of our construction f_{ineq} that computes INEQ and let $f^{(1)}_{\text{ineq}}(x)_i$ denote the ith output vector of the first layer. Our construction will be such that for $i > N/2$, on the ith input, the attention mechanism at the first layer will retrieve the $(i - N/2)$th input using tools described in Section 3.2.3. With the MLP, the first layer $f^{(1)}_{\text{ineq}}(x)_i$ will compute $x_i \oplus x_{i-\frac{N}{2}}$ for each $i > N/2$ where $x_i \oplus x_{i-\frac{N}{2}} = 0$ if $x_i = x_{i-\frac{N}{2}}$ and is 1 otherwise. The second layer will then take an OR over all those values which will result in computing $\text{INEQ}(x)$.

Proof. Let $x = (x_1, \ldots, x_N)$ denote our input vector. The input to the Transformer model will include the positions as well $\tilde{x} = ((x_1, 1), \ldots, (x_N, N))$. Let $\mathbf{x}_i \in \mathbb{R}^d$ denote the embedding of the ith input $\tilde{x}_i$ where $d = 2 + 2k$. Here, $k = O(\log N)$ is the dimension of vectors $\mathcal{T}(1), \ldots, \mathcal{T}(N)$ described in Corollary 0.1. The input embeddings will contain four parts and will be of the following form

$$\mathbf{x}_i = \begin{cases} [x_i, 0, \mathcal{T}(i), \mathcal{T}(i), 1] & \text{if } i \leq N/2, \\ [x_i, 0, \mathcal{T}(i), \mathcal{T}(i - \frac{N}{2}), 1] & \text{otherwise.} \end{cases}$$

The query vectors $Q(\mathbf{x}_i) = \mathbf{x}_i W_Q$ will be the last part of the embeddings, i.e., $Q(\mathbf{x}_i) = [\mathcal{T}(i), 1]$ for $i \leq N/2$ and is $\mathcal{T}(i - N/2)$ for $i > N/2$. Similarly, the key vectors $K(\mathbf{x}_i) = [\mathcal{T}(i), -1/2]$ for all i. The value vector will be a d-dimensional vector $V(\mathbf{x}_i) = [0, x_i, \mathbf{0}_k, \mathbf{0}_k]$. The query, key, and value transformations can be implemented with block matrices containing zero and identity matrices similar to the one described in Section 3.3.

By construction, the dot products $A_{i,j} = \langle Q(\mathbf{x}_i), K(\mathbf{x}_j) \rangle = \langle \mathcal{T}(i - N/2), \mathcal{T}(j) \rangle - 1/2$ will be such that for $i > N/2$,

$$A_{i,j} = \langle Q(\mathbf{x}_i), K(\mathbf{x}_j) \rangle \begin{cases} \geq 1/4 & \text{if } j = i - \frac{N}{2}, \\ \leq -1/4 & \text{otherwise.} \end{cases}$$

For $i < N/2$, the dot products $A_{i,j}$ will be greater than $1/4$ if $i = j$ and will be less than $-1/4$ for $i \neq j$. If this was a Transformer with the hard-attention mechanism, then, note that $\text{Att}(X)_i = [0, x_i, \mathbf{0}_k, \mathbf{0}_k]$ for $i \leq N/2$ and $\text{Att}(X)_i = [0, x_{i-N/2}, \mathbf{0}_k, \mathbf{0}_k]$ for $i > N/2$. With residual connections or another attention-head, the output of the attention block will include the original input as well, which will lead to

$$\text{Att}(X)_i + \mathbf{x}_i = \begin{cases} [x_i, x_i, \mathcal{T}(i), \mathcal{T}(i), 1] & \text{for } i \leq N/2, \\ [x_i, x_{i-N/2}, \mathcal{T}(i), \mathcal{T}(i), 1] & \text{for } i > N/2. \end{cases}$$

Then a simple ReLU FFN can compute the XOR of the first two values of the output vector from the attention block. Hence, by construction, the output vector from the first layer $f^{(1)}_{\text{ineq}}(x)_i = [x_i \oplus x_{i-N/2}, \ldots]$ for $i > N/2$ and $[0, \ldots]$ for $i \leq N/2$.

Second layer computing OR. The second layer of the Transformer will compute the OR over the first coordinate of the input vector, which is quite straightforward. Let the query vector $Q(\mathbf{x}^{(1)}_i) = 0$. The value vectors will be of the form $V(\mathbf{x}^{(1)}_i) = [(x_i \oplus x_{i-N/2}), 0, \ldots, 0]$ for $i > N/2$ and they will be zero vectors by construction for $i \leq N/2$. Then, regardless of the keys, the dot products will be 0 for each position,

and hence

$$\text{Att}(X)_N^{(2)} = \frac{1}{N}\sum_{i=N/2}^{N}(x_i \oplus x_{i-N/2}).$$

If $\text{Att}(X)_N^{(2)} \geq \frac{1}{N}$, then the FFN can output 1 and it can output 0 otherwise.

Softmax attention. We now describe how to retrieve the $x_{i-N/2}$ values with softmax attention in the first layer of the model. The approach is slightly different from the one used in Section 3.3 and makes use of finite precision rounding.

Consider another construction that is identical to the construction described above with the exception that $\tilde{Q}(\mathbf{x}_i) = \eta Q(\mathbf{x}_i) = \eta[\mathcal{T}(i), 1]$. We show that for large enough η and $i > N/2$, the weight $\text{softmax}(A)_{i,j}$ will be so close to 1 for $j = i - N/2$ that it will be rounded to 1. Similarly, it will be rounded to 0 for $j \neq i - N/2$. For $i \leq N/2$, the weight will be rounded to 1 for $i = j$ and will be rounded to 0 otherwise.

Recall that the finite precision implementation is parameterised by a large constant K_c (c.f. Section 3.2.2). For $i > N/2$ and $j = i - N/2$, there exists an η such that,

$$\left|1 - \frac{\exp(\eta\langle Q(\mathbf{x}_i), K(\mathbf{x}_j)\rangle)}{\sum_{k=1}^{N}\exp(\eta\langle Q(\mathbf{x}_i), K(\mathbf{x}_k)\rangle)}\right| \leq \frac{1}{2N^{\mathrm{K}_c}}.$$

For such an η, the weight $\text{softmax}(A)_{i,j}$ will be rounded to 1.

$$1 \geq \frac{\exp(\eta\langle Q(\mathbf{x}_i), K(\mathbf{x}_j)\rangle)}{\sum_{k=1}^{N}\exp(\eta\langle Q(\mathbf{x}_i), K(\mathbf{x}_k)\rangle)} \geq \frac{\exp(\frac{1}{4}\eta)}{\exp(\frac{1}{4}\eta) + (N-1)\exp(-\frac{1}{4}\eta)} \geq 1 - \frac{1}{2N^{\mathrm{K}_c}}$$

$$\implies 2N^{\mathrm{K}_c}\exp(\frac{1}{4}\eta) \geq (2N^{\mathrm{K}_c} - 1)(\exp(\frac{1}{4}\eta) + (N-1)\exp(-\frac{1}{4}\eta))$$

$$\implies \exp(\frac{\eta}{4}) \geq (N-1)(2N^{\mathrm{K}_c} - 1)\exp(-\frac{\eta}{4})$$

$$\implies \eta \geq 2\log\Big((N-1)(2N^{\mathrm{K}_c} - 1)\Big)$$

Thus, for $\eta = \log N + \mathrm{K}_c \log 2N$, the softmax attention weight at $j = i - N/2$ will be rounded to 1. Similarly, one may verify that if $\eta \geq \mathrm{K}_c \log 2N$, then the weight $\text{softmax}(A)_{i,j}$ for $j \neq i - N/2$ will be less than $1/2N^{\mathrm{K}_c}$ and hence will be rounded to 0. Hence, for $\eta \geq \log N + \mathrm{K}_c \log 2N$, the softmax attention will behave like hard attention, and as described earlier the Transformer will compute the INEQ function. This completes the proof.

□

The scaling in the attention mechanism in the last part can also be implemented in almost exactly the same way as the construction for Theorem 1. Implementing it that way then requires the ReLU FFN to act as a threshold function. The scaling described above makes use of the finite precision setting to amplify the dot products to the point that it acts as hard attention.

Generalising this result, we find that two-layer Transformers with logarithmic width can express a more general class of Boolean functions: thresholds of k-sparse features, a class including functions such as Equality and Disjointness. Since such functions cannot be expressed by one-layer Transformers and recurrent models with width $o(N)$, these results imply a *separation* on Equality and Disjointness: these functions can be expressed by small-sized two-layer Transformers, whereas one-layer Transformers and recurrent models must grow linearly with input length to represent them.

3.5.2 Implementing the Nearest Neighbour Algorithm

The goal of the nearest neighbour task (NstNb) is to analyse whether a sequence modelling architecture can implement the well-known nearest neighbour algorithm to make predictions.

Nearest Neighbours. In the NstNb task, a model is provided with a sequence of vectors and labels $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$ where $k \in \{\frac{N}{2}+1, \ldots, N\}$, the input *unit* vectors $\mathbf{x}_i \in \mathbb{R}^d$ and labels $y_i \in \{0, 1\}$. For each $\mathbf{x}_k$ where $k > N/2$, the output is the label corresponding to the nearest neighbour in $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$, that is, if $j = \arg\max_{i\in[k-1]} \mathbf{x}_k^\top \mathbf{x}_i$ or $j = \arg\min_{i\in[k-1]} \|\mathbf{x}_k - \mathbf{x}_i\|_2$, then the output for $\mathbf{x}_k$ is the label y_j. Since we are working with unit vectors, maximising the inner product is equivalent to minimising the ℓ_2 distance. If the second half of the sequence $\mathbf{x}_{\frac{N}{2}+1}, \ldots, \mathbf{x}_N$ is a permutation of the first half $\mathbf{x}_1, \ldots, \mathbf{x}_{\frac{N}{2}}$ then the task reduces to the Multi-Query Associative Recall (Mqar) task [10].

Assumptions. We will make two assumptions about the problem.

- **Norm.** All input vectors $\mathbf{x}_i$ are of unit norm, i.e., $\|x\|_2 = 1$.
- **Margin.** Existence of a margin between the dot product with the nearest neighbour and the dot product with other input vectors, i.e. there exists $\gamma \geq N^{-c}$ for some universal constant c, such that for any $k \in \{\frac{N}{2}+1, \ldots, N\}$, if $j^* = \arg\max_{i\in[k-1]} \mathbf{x}_k^\top \mathbf{x}_i$, then $\mathbf{x}_k^\top \mathbf{x}_{j^*} \geq \mathbf{x}_k^\top \mathbf{x}_i + \gamma$ for any $i \neq j^*$.

Relation to Associative Recall. The nearest neighbour task is closely related to the single and multi-query associative recall (Mqar) task introduced in Ba et al.

[13] and Arora et al. [10]. In the associative recall task, a model receives a sequence $s_1, y_1, \ldots, s_{k-1}, y_{k-1}, s_k$ where the symbols s_i belong to an alphabet $|\Sigma|$. Assume the symbols $s_1, \ldots, s_{k-1}$ to be distinct and the labels $y_i \in \{0, 1\}$ for simplicity. The query input s_k is a repetition of one of the preceding inputs. The goal of the model is to predict the label of the query input s_k by finding the exact match from the context and producing the corresponding label. A model that can compute the nearest neighbour algorithm can perform these associative recall tasks by embedding the sequence of symbols $s_1, y_1, \ldots, s_{k-1}, y_{k-1}, s_k$ as a sequence of vectors $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$. If a model receives a sequence $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$ where the vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{k-1}$ are distinct and the query vector $\mathbf{x}_k$ is a repetition then the task of applying nearest neighbour to predict the label for the query vector reduces to the single query associative recall task. Implementing the nearest neighbour algorithm is equivalent to finding the exact match for the query vector $\mathbf{x}_k$ and producing the label y_k corresponding to that input. In the case where the models are required to predict iteratively for all $\mathbf{x}_k$ for $k = \frac{N}{2} + 1, \ldots, N$, the task becomes equivalent to MQAR.

The MQAR task is a simplified version or a subset of the nearest neighbour task where the model is first provided with the prompt $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{N/2}, y_{N/2})$ and the subsequent $N/2$ input vectors are a permutation of the first $N/2$ vectors. In other words, for $N/2 < k \leq N$ and a sequence $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$, the model has to find the exact match of $\mathbf{x}_k$ in the first $N/2$ vectors and output the corresponding label. It is straightforward to see that any model that can perform the nearest neighbour task can also perform the MQAR task. Assume the size of the alphabet $|\Sigma| = \Theta(N)$. The embedding of each symbol $s_i \in \Sigma$ can be a distinct vector from the hypercube $\{-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\}^d$ where $d = \lceil \log |\Sigma| \rceil = O(\log N)$. The embeddings can be thought of as the normalised binary encodings of each symbol. Thus, a model receives a sequence of vectors $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$ corresponding to a sequence of symbols $s_1, y_1, \ldots, s_{k-1}, y_{k-1}, s_k$. Since the query vectors $\mathbf{x}_k$ for $k > N/2$ are repetitions, there is an exact match for each of them in $\mathbf{x}_1, \ldots, \mathbf{x}_{N/2}$. Suppose the exact match for a query $\mathbf{x}_k = \mathbf{x}_{j^*}$, then their dot products $\langle \mathbf{x}_k, \mathbf{x}_{j^*} \rangle = 1$ and the dot product with every other vector $\langle \mathbf{x}_k, \mathbf{x}_i \rangle \leq 1 - \frac{1}{\log N}$. Since the margin between the dot product with the nearest neighbour and other vectors satisfy $\frac{1}{\gamma} = O(\log N)$ and all the input embeddings have unit norms, the problem satisfies the assumptions of the nearest neighbour task.

Result. The following is one of our main results, which states that two-layer Transformers of logarithmic size can implement the nearest neighbour algorithm in their forward pass and, as a corollary, can also perform associative recall tasks like MQAR.

Recall that we say a model operates over p bits if all the weights and intermediate computations can be represented with p bits of precision.

Theorem 7. *For any $N \in \mathbb{N}$, there exists a 2-layer Transformer $f_{NN} \in TF^2_{m,p,2}$ with width $m = O(\log N)$ and precision $p = O(\log N)$ such that f_{NN} computes the nearest-neighbour task on all sequences of length at most N satisfying the assumptions above.*

For $N/2 < k \leq N$, the model will be provided the sequence $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$ where $\mathbf{x}_i \in \mathbb{R}^{d'}$ vectors contain the input points and y_is contain the labels. For clarity, we will refer to the embedding of all inputs as $\mathbf{z}_i \in \mathbb{R}^d$ where $\mathbf{z}_{2i-1} = \phi(\mathbf{x}_i, 2i-1)$ is the embedding of input points for $i = 1, \ldots, k$. Similarly, the vectors $\mathbf{z}_{2i} = \phi(y_i, 2i)$ will contain the embedding for the corresponding labels. Hence, technically, the sequence of input vectors to the Transformer model will be $(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{2k-1})$.

Overview of Idea. The explicit construction is a bit tedious, but the key ideas are straightforward to understand. The construction itself does not rely on input sequences of fixed lengths, unlike the one for the Equality problem.

Recall that for an input sequence $(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{2k-1})$, the odd indices $(2i-1)$ correspond to the inputs $\mathbf{x}_i$ and the even indices $(2i)$ contain the label information y_i. The final input $\mathbf{z}_{2k-1}$ contains the query input $\mathbf{x}_k$, and the goal of the model is to find the nearest neighbour input $\mathbf{x}_{j^*}$ and output the label corresponding to that y_{j^*}.

Intuitively, a two-layer Transformer can do that in the following way: the first layer can find the nearest neighbour $\mathbf{x}_{j^*}$ and retrieve the position of the label y_{j^*}. The second layer can then attend over that position and produce the desired label.

Challenges. There are a few challenges to executing this strategy, which our construction will address. If the input embeddings were identical to the input vectors, then $\mathbf{x}_k$ would have maximum dot product with itself and not with its nearest neighbour $\mathbf{x}_{j^*}$. Second, if you remove that, even then the dot product with some label vector could be larger than the dot product with the nearest neighbour (which could be close to -1). Lastly, suppose by design you have the maximum dot product with the desired input, you still need to retrieve the required information in a useful way, since we are working with softmax attention, which will also put weight over other inputs.

Key ideas. One can think of the vectors $\mathcal{T}(1), \ldots, \mathcal{T}(N)$ as some form of positional or address vectors of dimension $O(\log N)$. All vectors $\mathbf{z}_{2i-1}$ corresponding to inputs $\mathbf{x}_i$ will have the address vectors of their next position $\mathcal{T}(2i)$. Along with that, all the vectors $\mathbf{z}_i$ will also have their own address/position vectors $\mathcal{T}(i)$. If in the first layer, the model can attend over the vector $\mathbf{z}_{2j^*-1}$ with a high attention weight, then it will

be able to retrieve the required address vector $\mathcal{T}(2j^*)$ which it can then use in the second layer to retrieve the required label.
The input embeddings are designed in such a way that the maximum dot product will be with the desired input vector $\mathbf{x}_{j^*}$ or, more specifically, $\mathbf{z}_{2j^*-1}$. The embeddings are such that the dot products between the query vector of input $\mathbf{z}_{2k-1}$ and the key vectors of all other input vectors $\mathbf{z}_i$ will be of the following form,

$$\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle = \langle \mathbf{x}_k, \mathbf{x}_i\rangle - c_1\langle \mathcal{T}(2k-1), \mathcal{T}(2i-1)\rangle \quad \text{for } i = 1, \ldots, k.$$

See that for $i \neq k$, $\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle \approx \langle \mathbf{x}_k, \mathbf{x}_i\rangle$ whereas for $i = k$, $\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle < -2$ or much smaller based on the constant c_1. Hence, attention weight will be smaller on the query input itself compared to other inputs.
Secondly, for $i = 1, \ldots, k-1$, the dot products with the label vectors will be of the following form,

$$\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i})\rangle = \langle \mathbf{x}_k, \mathbf{0}\rangle - c_1\langle \mathcal{T}(2k-1), \mathcal{T}(2i-1)\rangle - c_2 \approx -c_2 \quad \text{for } i = 1, \ldots, k.$$

which will be small depending on the constant c_2. Hence, the dot product will be maximum with input with the nearest neighbour $\mathbf{x}_{j^*}$ with a margin $\Omega(\gamma)$.
To retrieve and use $\mathcal{T}(2j^*)$ in the next layer, we do not need to hard-attend on $\mathbf{z}_{2j^*-1}$. Since the address vectors are of the form $\mathcal{T}(i) \in \{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}^k$, we only retrieve the corresponding sign vectors $\mathcal{T}_s(i) = \sqrt{k}\mathcal{T}(i) \in \{-1, 1\}^k$. Since the attention weight will be maximum on $\mathbf{z}_{2j^*-1}$ with a margin, we can scale the query vectors to increase the weight to a sufficient value and then use ReLU as a form of threshold to obtain $\mathcal{T}_s(2j^*)$. The second layer is then straightforward and will retrieve the desired label.

Proof. We will now describe the explicit construction for Transformers to compute nearest neighbours.
Input Embeddings. The embedding vectors are defined in the following way,

$$\phi(\mathbf{x}_i, 2i-1) = \mathbf{z}_{2i-1} = [\mathbf{x}_i, 0, 1, \mathcal{T}(2i-1), \mathcal{T}(2i), \mathbf{0}_k] \tag{3.6}$$

$$\phi(y_i, 2i) = \mathbf{z}_{2i} = [\mathbf{0}_{d'}, y_i, -2, \mathcal{T}(2i), \mathcal{T}(2i), \mathbf{0}_k].$$

Here, the vectors $\mathcal{T}(i) \in \{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}^k$ are JL transformations as described in Section 3.2.3 and hence $k = O(\log N)$. The dimension of the embedding vectors $d = 3k + d' + 2 = O(\log N)$. The vectors $\mathcal{T}(1), \ldots, \mathcal{T}(2N)$ are such that,

$$\langle \mathcal{T}(i), \mathcal{T}(j)\rangle = \begin{cases} 1 \pm \gamma/100 & \text{if } i = j, \\ 0 \pm \gamma/100 & \text{otherwise.} \end{cases}$$

Query, key and value vectors. The query and key transformations in the first layer are designed in the following way,

$$Q(\mathbf{z}_{2k-1}) = [\mathbf{x}_k, 1, -10\mathcal{T}(2k-1)]$$

$$K(\mathbf{z}_{2i-1}) = [\mathbf{x}_i, 3, \mathcal{T}(2i-1)] \quad \text{for } i = 1, \ldots, k,$$

$$K(\mathbf{z}_{2i}) = [\mathbf{0}_d, -2.3, \mathcal{T}(2i)] \quad \text{for } i = 1, \ldots, k.$$

The value vectors will only have the 5th part ($\mathcal{T}(2i)$) in the last slot, and all other values will be 0. Let $\mathcal{T}_s(i) = \sqrt{k}\mathcal{T}(i) \in \{-1, 1\}^k$, that is, $\mathcal{T}_s(i)$ contains the signs of the vector $\mathcal{T}(i)$. The value vector will be of the form $V(\mathbf{z}_{2i-1}) = [\mathbf{0}_{d'}, 0, 0, \mathbf{0}_k, \mathbf{0}_k, 2\mathcal{T}_s(2i)]$ and $V(\mathbf{z}_{2i}) = [\mathbf{0}_{d'}, 0, 0, \mathbf{0}_k, \mathbf{0}_k, 2\mathcal{T}_s(2i)]$ for all $i = 1, \ldots, N$. The goal of the attention in the first layer is to retrieve the vector $\mathcal{T}_s(2j^*)$ corresponding to the label of the nearest neighbour.

Inner products. The design of such query and key transformation ensures that for an input query $\mathbf{x}_k$, the dot product is maximum with its nearest neighbour $\mathbf{x}_{j^*}$ and not with itself or any of the embeddings of the labels y_is.

The inner products $A_{2k-1,2i-1} = \langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle$ have the following form,

$$\begin{aligned}\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle &= \langle \mathbf{x}_k, \mathbf{x}_i\rangle + 2 - 10\langle \mathcal{T}(2k-1), \mathcal{T}(2i-1)\rangle \\ &= \begin{cases} \langle \mathbf{x}_k, \mathbf{x}_i\rangle + 3 \pm \gamma/10 & \text{if } i \neq k, \\ \langle \mathbf{x}_k, \mathbf{x}_i\rangle + 3 - 10 \pm \gamma/10 & \text{if } i = k. \end{cases} \\ \implies \langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle &\begin{cases} \geq 1 & \text{if } i \neq k, \\ \leq -5 & \text{if } i = k. \end{cases}\end{aligned}$$

Similarly, the inner product with the label vectors $\mathbf{z}_{2i}$s is less than 0 for all i as well,

$$\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i})\rangle = 0 - 6 \pm \gamma/10 \leq -5$$

To summarise, the query and key vectors are such that for the query input $\mathbf{z}_2k - 1$, the dot product with itself $\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2k-1})\rangle \leq -5$ and the dot product with all label vectors containing y_i is $\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i})\rangle \leq -5$. The remaining dot products are with the vectors of interest, which include the nearest neighbour input point,

$$\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle = \langle \mathbf{x}_k, \mathbf{x}_i\rangle \pm \gamma/10 \quad \text{for } i = 1, \ldots, k-1.$$

Suppose $j^* = \arg\max_{i\in[k-1]} \mathbf{x}_k^T \mathbf{x}_i$ is the index for the input point which has the minimum L2 distance or maximum inner product with the query vector $\mathbf{x}_k$. Then we have that,

$$\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2j^*-1})\rangle - \langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle \geq \gamma - \gamma/5 \tag{3.7}$$

for all $i \neq j^*$. This indicates the maximum inner product will be with its nearest neighbour and all other inner products will have a margin of at least $\tau = \frac{4}{5}\gamma$.

If we have a Transformer with a hard-attention mechanism, then it will attend only to the input which is the nearest neighbour of the query vector $\mathbf{x}_k$. However, with softmax attention, it is non-trivial to only attend over a single input.

The embedding of all input vectors $\mathbf{x}_i$ contains a vector $\mathcal{T}(2i)$ (see Eq. 3.6) which serves as a key vector to retrieve the label following that input vector. Note that we only need to retrieve the signs of the vector $\mathcal{T}(2i)$ since the vectors are of the form $\{-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\}^k$. We can then use it to retrieve the label of the corresponding input in the next layer.

Retrieving with softmax. We show using softmax attention and a ReLU FFN we can retrieve the required $\mathcal{T}(2j^*)$. In particular, we will obtain $\mathcal{T}_s(2j^*) = \sqrt{k}\mathcal{T}(2j^*) \in \{-1, 1\}^k$. As shown earlier, if $j^* = \arg\max_{i\in[k-1]} \mathbf{x}_k^T\mathbf{x}_i$, then the maximum dot product $\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2j^*-1})\rangle \geq \langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2i-1})\rangle + \tau$ is greater by a margin $\tau = \frac{4}{5}\gamma$. The value vector corresponding to $\mathbf{z}_{j^*}$, i.e., $V(\mathbf{z}_{j^*}) = [0, \ldots, 0, 2\mathcal{T}_s(2j^*)]$ has the key vector which will allow the next layer to retrieve the required label.

The basic idea is that even if at least 9/10 of the attention weight is on $V(\mathbf{z}_{j^*})$ and essentially $2\mathcal{T}_s(2j^*)$ then that suffices to preserve the signs using a ReLU FFN.

Let $\sigma(a)$ be a function such that $\sigma(a) = 1$ for $a > 1$, $\sigma(a) = -1$ for $a < -1$, and $\sigma(a) = a$ for $-1 \leq a \leq 1$. See that $\sigma(a)$ can easily be implemented by a ReLU FFN since it is essentially $\text{ReLU}(x + 1) - \text{ReLU}(x - 1) - 1$.

Without loss of generality, let's say $2\mathcal{T}_s(2j^*)_1 = +2$. If the attention weight on it is 9/10 and the remaining 1/10 weight is distributed among the rest of the inputs, then in the worst case, that value will be $\frac{9}{10}2 - \frac{1}{10}2 \geq 1$. Hence, applying $\sigma(\cdot)$ over the value will result in $+1$. The same goes for the case when $2\mathcal{T}_s(2j^*)_1 = -2$ and for other indices of $2\mathcal{T}_s(2j^*)$. Thus, it suffices to show that the attention weight over $V(\mathbf{z}_{2j^*-1})$ can be greater than 9/10 since it implies that with the ReLU FFN, the output for $\mathbf{z}_{2k-1}$ in the first layer will contain $\mathcal{T}_s(2j^*)$.

Consider another construction which is identical to the construction described above with the exception that $\tilde{Q}(\mathbf{z}_i) = \eta Q(\mathbf{z}_i)$. We show that for large enough η, the weight $\text{softmax}(A)_{2k-1,2j^*-1}$ will be greater than 9/10 and the remaining weights combined will be less than 1/10. Let $\beta = \langle \tilde{Q}(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2j^*-1})\rangle$ and $Z \in \mathbb{R}^{N\times d}$ contain vectors $(\mathbf{z}_1, \ldots, \mathbf{z}_{2k-1})$. Then, for any $r \in \mathbb{N}$,

$$\begin{aligned}
&\text{softmax}(\tilde{Q}(\mathbf{z}_{2k-1})K(Z)^T)_{2j^*-1} \\
&= \frac{\exp(\eta\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2j^*-1})\rangle)}{\exp(\eta\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_{2j^*-1})\rangle) + \sum_{p\neq 2j^*-1}\exp(\eta\langle Q(\mathbf{z}_{2k-1}), K(\mathbf{z}_p)\rangle)} \\
&\geq \frac{\exp(\eta\beta)}{\exp(\eta\beta) + (2k-1)\exp(\eta(\beta-\tau))} \geq \frac{\exp(\eta\beta)}{\exp(\eta\beta) + 2N\exp(\eta(\beta-\tau))} \geq \frac{r-1}{r} \\
&\implies \exp(\eta\beta) \geq (r-1)(2N)\exp(\eta(\beta-\tau)) \\
&\implies \eta \geq \frac{1}{\tau}\log(r-1)2N.
\end{aligned}$$

Hence, for $r = 10$, if $\eta = \frac{5}{4\gamma}\log 18N$ then the attention weight on the $2j^* - 1$th input vector will be greater than $9/10$. Similarly, one can verify that for $\eta = \frac{5}{4\gamma}\log 18N$, the attention weight for the rest of the inputs combined is at most $1/10$.

Output of the first layer. Let $\mathbf{z}_i^{(1)}$ denote the output of the first layer on the ith input vector. By construction, with MLP and residual connection after the attention block, the output of the first layer is such that,

$$\begin{aligned}
\mathbf{z}_{2k-1}^{(1)} &= [\mathbf{x}_k, 0, 1, \mathcal{T}(2k-1), \mathcal{T}(2k), \mathcal{T}_s(2j^*)] \\
\mathbf{z}_{2i-1}^{(1)} &= [\mathbf{x}_i, 0, 1, \mathcal{T}(2i-1), \mathcal{T}(2i), \ldots] \quad \text{for } i = 1, \ldots, k-1 \\
\mathbf{z}_{2i}^{(1)} &= [\mathbf{0}_{d'}, y_i, -2, \mathcal{T}(2i), \mathcal{T}(2i), \ldots] \quad \text{for } i = 1, \ldots, k-1.
\end{aligned}$$

Second Layer. Since we have the address/key vector $\mathcal{T}_s(2j^*)$ for the target label as the input in the first layer, it is straightforward to apply attention and $\sigma(\cdot)$ with ReLU FFN to produce the desired label.

See that if the query vector $Q(\mathbf{z}_{2k-1}^{(1)}) = [\frac{1}{k}\mathcal{T}_s(2j^*)]$, and the key vectors contain the 4th part of the input vector, that is, $K(\mathbf{z}_i^{(1)}) = [\mathcal{T}(i)]$, then the dot product will be greater than $1 - \gamma/100$ with the desired input at $2j^*$ and will be less than $\gamma/100$ for the rest of the inputs. The value vectors will be assigned the second part of the input $V(\mathbf{z}_{2i}^{(1)}) = [\mathbf{0}_{d'}, y_i, 0, \ldots, 0]$ and will $V(\mathbf{z}_{2i-1}^{(1)}) = [0, \ldots, 0]$ for all $i = 1, \ldots, k$. Using the techniques used for the first layer, it is straightforward to see that a scaling of the query vector and applying $\sigma(\cdot)$ to the output will produce the desired label y_{j^*}.

□

Theorem 8. *Any recurrent model with a hidden state of width m with p bits of precision that can perform the nearest neighbour task for all inputs of length N must have $m \geq N/2p$.*

Proof. The proof is via a reduction from the disjointness problem. We show that the lower bound is true even for the restricted problem of multi-query associative recall (MQAR). Recall that in the MQAR task the sequence of query inputs $\mathbf{x}_{\frac{N}{2}+1}, \ldots, \mathbf{x}_N$ is a permutation of the sequence of labelled vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{N/2}$.

If there exists a recurrent model $\mathcal{R}$ that can solve the MQAR task, then we show that Alice and Bob can use it to follow a communication protocol and compute the DISJ function.

The communication protocol is as follows. Alice and Bob have two Boolean vectors $a, b \in \{0,1\}^{N/2}$ respectively. Both of them know the description of the recurrent model $\mathcal{R}$. Alice and Bob have decided on a set of $N/2$ vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{N/2}$ in $\mathbb{S}^{d-1}$ that they will use in the communication protocol. Choosing any $N/2$ distinct vectors from the hypercube $\{-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\}^d$ will suffice and also satisfy the assumptions mentioned in Section 3.5.2.

To reiterate, both Alice and Bob have the model parameters/description and have decided on a set of $N/2$ unit vectors as a part of their communication protocol. Alice then uses the recurrent model $\mathcal{R}$ and provides the sequence of pairs $(\mathbf{v}_i, a_i)$ in any arbitrary order. Alice then sends the hidden state vector $\mathbf{h}_N$ to Bob. Bob then uses the model $\mathcal{R}$ and iteratively provides $\mathbf{v}_i$ vectors along with the generated output in any order. Bob knows that the output produced by the recurrent model for the vector $\mathbf{v}_i$ corresponds to the value a_i. Hence, Bob obtains the entire vector a and computes $\text{DISJ}(a, b)$.

Since Alice sent the hidden state vector, which requires mp bits, and they could compute $\text{DISJ}(a, b)$ over $\{0,1\}^{N/2}$ by exchanging mp bits, by Fact 1 we have that $mp \geq N/2$. □

Discussion. Prior works [10] have empirically demonstrated that Transformer-based LLMs can exhibit mechanisms such as MQAR and also nearest neighbours, as we will see in Chapter 4. Further, on synthetic setups, they have observed that recurrent models struggle to perform these tasks compared to Transformers. Our results take a step towards understanding the differences in the performance between the two architectures.

3.5.3 Difficulty of Deriving Communication-based Lower Bounds for 2-layer Transformers

Our lower bounds both for RNNs and for one-layer transformers are based on communication complexity arguments. Here, we provide evidence that other techniques may

be needed to establish lower bounds for two-layer transformers by showing that, in a certain sense, no short communication protocol of the same kind as Theorem 4 can exist for Alice and Bob to obtain the output of any two-layer transformer. We start from the following lemma:

Lemma 4. *Assume N is even. For every partition S_A, S_B of $\{1, \ldots, N\}$ where $|S_A| = |S_B|$, there is a 2-layer transformer $f \in TF^2_{d,p,2}$ with width $m = O(\log N)$ with precision $p = O(\log N)$ with $f(x) \in \{0,1\}$ for each $x \in \{0,1\}^N$, such that Alice and Bob, having access to x_A and x_B respectively, need to exchange $\geq \frac{N}{2}$ bits to compute $f(x)$.*

Proof. For any partitioning S_A, S_B, consider the task of determining whether x_{S_A} and x_{S_B} are identical. That is, define (for $x \in \{0,1\}^N$):

$$f_{S_A,S_B}(x) = \begin{cases} 1 & \text{if } x_{S_A} \neq x_{S_B} \\ 0 & \text{else} \end{cases} \tag{3.8}$$

If Alice and Bob have access to x_{S_A} and x_{S_B}, respectively, they need to exchange $\geq \frac{N}{2}$ bits to compute $f(x)$. Now by Theorem 6, there is a transformer $f_{ineq} \in \mathrm{TF}^2_{m,p,2}$ that computes $f_{[1,\ldots,n/2],[n/2+1,\ldots,n]}$, where $m = O(\log N)$ and $p = O(\log N)$. Now renumbering the positional encodings results in a transformer computing f_{S_A,S_B}. □

Corollary 8.1. *If there is any communication protocol by which Alice and Bob can compute the output for any two-layer Transformer, for any partition, with $O(mpHg(n))$ bits, then it must be the case that $g(n) = \Omega\left(\frac{N}{(\log N)^2}\right)$.*

Proof. Suppose there is a communication protocol such that for any two-layer Transformer and over any partition, the protocol can compute the output of the Transformer with

$$o\left(mpH\frac{N}{(\log N)^2}\right) \tag{3.9}$$

bits. But by Lemma 4, for each partition, there exists a two-layer Transformer with

$$mpH = O((\log N)^2) \tag{3.10}$$

such that Alice and Bob need to exchange $\geq N/2$ bits to compute its output. We obtain a contradiction. □

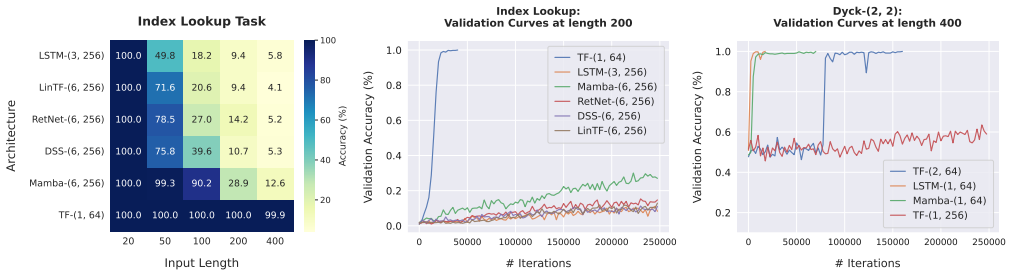


Figure 3.2: Performance of models on the Index Lookup and bounded Dyck task. Labels such as TF-(1, 64) denote Transformers with 1 layer and 64 widths. See Section 3.6 for more details.

3.6 Empirical Analysis

While we focus on the differences in the representational capabilities of Transformers and recurrent models, it is natural to examine if differences of a similar nature arise in their empirical performance. One thing to keep in mind is that positive results regarding expressiveness presented in earlier sections do not imply that models can *learn* such tasks. With regard to negative results, they do imply that when the sequence length is much larger than the size of the hidden state or width of the model, then the model will be incapable of representing the task and consequently fail to learn the task. However, even for one-layer recurrent models with hidden states of size 128 with 64 bits of precision, our lower bound applies at lengths over 8k.

In this section, we investigate the performance of Transformers and recurrent models on tasks such as Index Lookup and recognizing bounded Dyck languages on sequences of small lengths (< 1000). Our experiments are designed to answer the following questions: (1) Are one-layer Transformers better than larger recurrent models on the Index Lookup task? (2) Are recurrent models and two-layer Transformers better than one-layer Transformers at recognising the Dyck-(2, 2) language? Importantly, as our results concern the scaling of the model size with the input length, we are specifically interested in the behaviour of different models across input lengths.

We also explore the performance of models on string equality in Section 3.6.4. Tasks like MQAR have already been analysed empirically in prior works [10], so we do not include them. We empirically investigate the nearest neighbour task more comprehensively in Chapter 4 within the framework of in-context learning.

3.6.1 Setup and Results

Setup and Training details. We train the models with cross-entropy loss using the Adam optimiser [86]. The models are trained for up to 250k steps, where at each step we sample a fresh batch of 64 training examples – resulting in $\approx$ 16 million examples over 250k steps. The models are evaluated on 5000 examples for each task. For each model, we tune the various hyperparameters, notably across learning rates $\in \{\text{1e-2}, \text{5e-3}, \ldots, \text{1e-6}\}$ to find the best-performing model.

Index Lookup Task. We compare the performance of one-layer Transformers with five different recurrent models – LSTMs [73], state space models such as DSS [67] and Mamba [66], linear Transformers [82], and its variant RetNet [149]. We explore the performance across various lengths $\in \{20, 50, 100, 200, 400\}$. We evaluate relatively small-sized Transformers with widths 64 against recurrent models with up to 6 layers and widths or hidden state size of 256. The size of the alphabet in the experiments is $|\Sigma| = 64$. Figure 3.2 (left) depicts the performance of all models across various lengths and Figure 3.2 (middle) depicts the validation curves during training on examples of length 200. As depicted by the figures, while one-layer Transformers with width 64 achieve near-perfect accuracy within a few thousand steps, the performance of relatively larger recurrent or state-space models degrades on lengths over 100, and they fail to learn even with $10\times$ training iterations. We explore the influence of width on the performance of Mamba in Section 3.6.3.

Bounded Dycks. For DYCK-2 with depth at most 2, our separation results apply to one-layer Transformers and nonlinear recurrent models such as LSTMs but not to linear RNNs such as state-space models and linear Transformers. Hence, we are primarily interested in the difference in performance between one-layer Transformers and LSTMs. In our experiments, we compare the performance of one-layer Transformers with relatively smaller recurrent models such as LSTMs and two-layer Transformers. We also include Mamba for reference. We consider LSTMs and Mamba with hidden state sizes of 64 and, similarly, two-layer Transformers with width 64. We evaluate a one-layer Transformer with a width of 256 across lengths $\in \{20, \ldots, 400\}$ most of which are smaller than the width of the model. We observe that one-layer Transformers achieve near-perfect accuracy up to lengths 100 but struggle on higher lengths. In contrast, small-sized recurrent models as well as two-layer Transformers can achieve near-perfect accuracy for lengths up to 400. Figure 3.2 (right) depicts the validation curve of the models on examples of length 400.

3.6.2 Implementation Details

In this section, we discuss the details of implementation and data generation for experiments described in Section 3.6. Further, we discuss some additional experiments on the string equality task.

Implementation and hyperparameters. All of our implementations for experiments are based on PyTorch [121]. For our experiments with Transformers, we use the Huggingface Transformers library [171] with the GPT-2 backbone as well as our own custom implementation. We use PyTorch's standard or in-built implementation for experiments with LSTMs. For state-space models like Mamba [66] and Diagonal state-space models [67], we use the official implementation provided by the authors for our experiments. For RetNet [149] and Linear Transformers, we use our own custom implementation.

For each task, we tune the models across several hyperparameters and report the results based on the best-performing model. We use grid search to tune the models for each task. For each architecture excluding LSTMs, we tuned across depths $\in \{1, 2, 4, 6\}$ and widths $\{64, 128, 256, 512\}$. Since LSTMs with large depths are hard to train [120], we only consider LSTMs with depths up to 3. For Transformers and linear Transformer variants, we tune the heads across $\{4, 8\}$. For all models, we tune the learning rate across $\{0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.000001\}$. We primarily use Transformers with absolute encodings for the result presented in the chapter. We train the models for up to 250k steps unless they achieve (almost) perfect validation accuracy earlier. After training, we evaluate the models on a fresh set of 5000 examples.

Note, however, that the focus of our work is slightly different and for some problems, we are interested in the behaviour of small-sized (width or depth) models of one architecture and a relatively larger model for another architecture. For instance, in the index lookup task, we are concerned with how well one-layer Transformers with small widths fare against relatively larger recurrent models. Additionally, our focus is on the comparison of different architectures of fixed sizes across lengths and not primarily on how well models of the scale used in practice perform these tasks on much larger lengths. Hence, we do not consider models with much larger depth or width.

Compute. All our experiments were conducted using 8 NVIDIA Tesla V100 GPUs each with 16GB memory and 16 NVIDIA GTX 1080 Ti GPUs each with 12GB memory. Each run for 500k steps could take between 1 hour and 16 hours depending on the length of the inputs and the size of the models. Some runs are much shorter if the

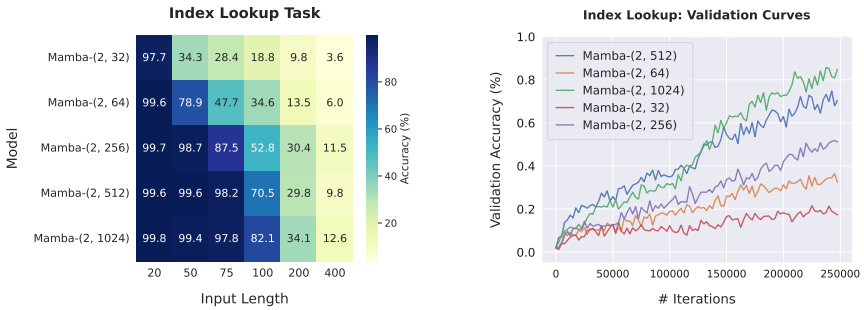


Figure 3.3: Performance of Mamba on the Index Lookup task across various lengths and widths. See Section 3.6.3 for more details.

model achieves high accuracy quite early (e.g. on lengths 20). While each individual run does not take a significant amount of time, tuning the models, particularly for the tasks where they fail to learn requires several runs and consequently a much longer duration. We estimate that all the runs across hyperparameters, architectures, and input lengths took ≤ 1200 GPU hours in total on the GPUs mentioned above.

3.6.3 Additional Experiments and Data Generation

Index Lookup Task. The input sequences are comprised of symbols in Σ and a positional token in $[N]$. In our experiments, the size of the set of symbols is $|\Sigma| = 64$ and N varies from 20 to 400. To create each example, we first sample a length k uniformly between 10 and N (20 - 400) and then sample N symbols independently from Σ uniformly at random. Lastly, we sample a position between 1 and k uniformly at random and append it to the sequence to produce a labeled example for training or evaluation. During evaluation, the model is tested on sequences of length exactly N.
Bounded Dycks. For the Dyck-(2, 2) task, we generate the examples in such a way that with 0.5 probability, the generated string is well-balanced with depth at most 2, and with 0.5 probability it does not belong to the language and has label 0. The generated strings are of length exactly N in both cases. To generate positive examples we use the following strategy: we iterate over $N-2$ steps and for each step, we check whether the current depth of the stack is < 2. If the depth of the stack is 2, the sequence continues with the closing bracket corresponding to the open bracket in the stack. If the depth is < 2, the sequence is either continued by choosing one of the open brackets uniformly if the stack is empty or by choosing uniformly between open brackets and the closing bracket if the stack is non-empty. After $N-2$ steps, the generated string could be a well-balanced string of length $N-2$ in which case we add a depth 1 string of length 2 at the end which results in a well-balanced string of

length N. Otherwise, the stack could be nonempty, so we add the remaining closing brackets, which also leads to a string of length N.

To generate negative examples, we first sample a positive example and then corrupt some of the symbols in the string. For strings of length N, we first sample a number $k = 1, \ldots, N/10$ uniformly at random and then pick k different indices uniformly at random. For each of those positions, with probability $1/2$, we swap the types of brackets, e.g. round '(' to square '[', and with probability $1/2$ we switch open brackets to closing brackets (or vice versa). There is a very small probability that after the corruption the resulting string will still be a valid well-balanced string of depth at most 2, in which case we redo the corruption again. The probability of the event is too low to affect the efficiency of the generation process.

Additional Experiment with Mamba. We explore how the size of the hidden state influences the performance of a Mamba model across various lengths on the Index Lookup task. We evaluate two-layer models of different widths $\{32, 64, 256, 512, 1024\}$ across various lengths ranging from 20 to 400. We find a clear trend where the performance of Mamba models increases monotonically with the increase in the width of the model (see Figure 3.3). While the performance does exactly scale linearly with width it is still somewhat interesting that the trend exists. We did a similar experiment with LSTM but did not observe such a trend and for lengths above 100 the performance remained at chance level even when the width was increased. Note, however, that even with a width of 1024, the performance of Mamba is still much worse than a one-layer Transformer with a width of 64.

3.6.4 String Equality Task

We explore a few different strategies to evaluate the performance of models on the string equality task. In these experiments, the goal of the models is to determine whether the first half of the input string and the second half of the string are equal. The first two experiments are in the standard classification setting but differ in the way the negative examples are created. The third experiment is in the next character prediction which is commonly used in prior works to test models on formal languages. For the equality task, we find that it is not straightforward to generate negative examples in a way that the problem cannot be solved using shortcuts. Hence, we include the next character prediction setting which does not require the generation of negative examples.

Summary of results. The results on the String Equality task are relatively more nuanced than the results for index lookup and DYCK-2. On a standard binary

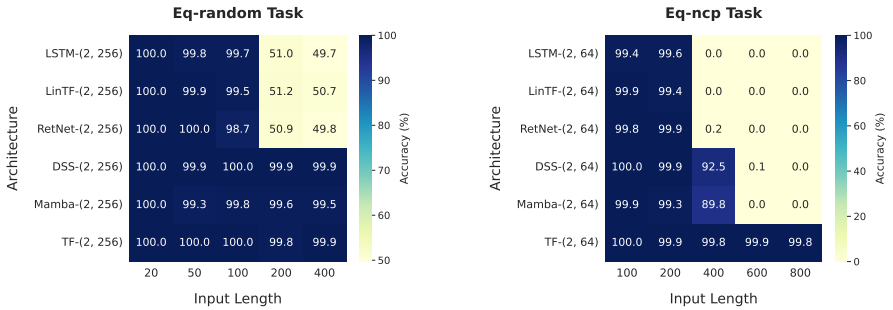


Figure 3.4: Performance of architectures on the Equality task. See Section 3.6.4 for more details.

classification setup, one must make a design choice regarding the generation of negative examples that could influence the difficulty of the task. We first conduct experiments in the classification setup by generating negative examples using two different strategies. On one task, we find that while recurrent models like LSTMs struggle beyond a certain length, state-space models like DSS and Mamba are able to match Transformer's performance. In the second task, we find that all models are able to achieve near-perfect accuracy. We note that both classification tasks can be solved by using shortcuts based on the way the negative examples are created. Hence, we explore another strategy called the next character prediction setting inspired by prior works [150, 59] on empirical analysis on formal languages. We note that the task in that setting becomes almost identical to the copying task where a model observes a sequence and then has to produce the same sequence. In that setting, we observe that at a small width like 64, state-space models fail to perform the task accurately at certain lengths whereas Transformers with the same width succeed. For models with larger widths, we find that DSS and Transformers succeed at performing the tasks for the lengths considered in our experiments.

Setup. In all our experiments with string equality, we have a vocabulary Σ which contains one token that is reserved as a separator token '[SEP]' and is placed between the first half and the second half of the input string. For each task, while creating an example we first pick the length k uniformly at random from $\frac{N}{10}, \frac{N}{10}+2, \frac{N}{10}+4, \ldots N$. The choice of lengths is due to the requirement that the length of the strings must be even. During evaluation, we test the models on strings of length exactly N.

(i) Eq-random. The first experiment is pretty straightforward and contains binary strings, i.e., $\Sigma = \{0, 1, [\text{SEP}]\}$. With probability $1/2$, we create a positive example by first sampling a string of length $k/2$ uniformly followed by the separator token and then the same string again. By construction, the created example has a positive label.

With probability 1/2, we create the second half of the string by sampling another string of the same length where each symbol is sampled uniformly at random. We assign the label based on whether or not it is equal to the first half but with high probability $(1 - \frac{1}{2^{k/2}})$, the example has a negative label.

(ii) Eq-one. In the second experiment, we have a vocabulary with 1024 and we create the positive example in the same way as earlier in the Eq-one setting by sampling strings uniformly at random. To generate a negative example, we set the second half to be equal to the first half and then change the symbol at only one of the positions. In other words, the second half matches the first half at all positions except one.

The training details and hyperparameters are almost identical to the experiments described earlier.

Results. On both the Eq-random task and Eq-one task we find that Transformers achieve near-perfect accuracy on lengths up to 400. Recurrent models like LSTMs and linear Transformers struggle at lengths beyond 100 on the Eq-random task. On the other hand, we find that recently proposed state-space models like DSS and Mamba are able to match Transformers' performance on both tasks (See Figure 3.4 left). On the Eq-one task, we find that for lengths up to 400, all models are able to achieve near-perfect accuracy.

Remark. One thing to note is that both of the experimental setups described above can be solved using some form of shortcuts. In the first case, it is straightforward to see that the first half and the second half will differ at about half the positions on average. A recurrent algorithm does not necessarily have to store the first $N/2$ elements to compute the output correctly with high probability. Even if it stores the first few elements, then with high probability, it can determine the label of the string correctly. For the second case (Eq-one), even if it might seem difficult at first look, a recurrent model can solve it perfectly by just maintaining a dictionary of the counts of each symbol in the vocabulary. Since the first half and the second half differ at exactly one position, the number of occurrences of at least one symbol will be different in the two halves.

To rule out such phenomena, we adopt the next symbol/character prediction setting [59, 133, 150, 51].

(iii) Eq-ncp. In the next character setting (NCP), a model is required to predict the next set of valid continuations for every prefix of a given string. It can be seen as a multi-label classification problem for every prefix of a given string. The prediction for a particular input string is considered correct if the prediction for every prefix is correct.

In the context of the string equality task with length N, for the first $N/2$ symbols, all symbols are valid continuations, and hence the predictions for the first half of the string are trivial. After observing the separator token [SEP], only one of the $|\Sigma|$ symbols is allowed at every prefix until the end of the string. We note that the prediction problem becomes equivalent to copying [79] a sequence of symbols. For our experiments the vocabulary size $|\Sigma| = 1024$. We explore sequences of higher lengths up to 800. In this setting, we find that when the model sizes are restricted, i.e., the width of the models is 64, state-space models such as DSS and Mamba struggle to perform better than chance-level accuracy for lengths over 400. In contrast, Transformers are able to achieve near-perfect accuracy (See Figure 3.4). However, unlike the case of Index Lookup, we find that the DSS architecture in particular is able to solve the task for lengths up to 800 with larger widths in the NCP setting.

Discussion. We discuss a few takeaways from our experiments. For the Index Lookup task, our results indicate that even small-sized one-layer Transformers can learn a lot more efficiently than recurrent models of much larger sizes. The experiments with bounded Dycks are primarily for one-layer Transformers and indicate that they learn at a much slower rate than recurrent models like LSTMs and even two-layer Transformers. On the string equality task, the difference in performance between Transformers and recurrent models is not as stark as the Index Lookup task, particularly with state-space models such as DSS. However, unlike Transformers, they seem to struggle on long sequences in the NCP setting when the widths of models are small.

3.7 Discussion and Final Remarks

Based on prior theoretical results, it is known that, while recurrent models can express any regular language [88, 87], Transformers with logarithmic precision can only express languages in the class of uniform constant depth threshold circuits (TC^0) [101]. These results indicate that—under standard conjectures—Transformers are unable to represent certain state-tracking tasks that recurrent models can represent. With such results, it might appear that Transformers are less expressive than recurrent models–potentially at odds with the persistent practical success of Transformer-based LLMs. Our findings, however, show that when the model size is constrained relative to the sequence length, a variety of tasks relevant to practice can be represented by small-sized Transformers but not by recurrent models. Our results suggest that the attention mechanism does lead to expressiveness that cannot be replicated by recurrent architectures, even with arbitrary transition functions.

Limitations. A general limitation of this line of work is that positive expressivity results do not imply that the problems under consideration are learnable. Additionally, while lower bounds for an architecture imply difficulty in learning, when using double precision, these results only apply to very long sequences in practice. Our results (and probably techniques) do not imply any limitations on two-layer Transformers; this is left as an open question. We note that communication complexity-based techniques akin to Theorem 4 cannot exist for two-layer Transformers (cf. Section 3.5.3). Hence, we believe that other tools will be needed to prove lower bounds for two-layer Transformers.

Chapter 4

Transformers as Learning Algorithms and In-Context Learning

4.1 Introduction

While Chapter 3 primarily focused on the expressivity of models, in this chapter, we will empirically investigate various aspects of Transformers and other architectures to learn more about in-context learning.

Transformer-based large language models (LLMs) have shown a remarkable ability to learn tasks in-context using a handful of demonstrations and without updating their weights [28]. Demystifying this 'in-context learning' ability theoretically and empirically is an intriguing direction. Since real-world language data and tasks are hard to define precisely, in-context learning has been studied in various simplified yet well-defined settings [32, 69].

Recently, [57] proposed a framework to understand the in-context learning phenomenon, wherein the model receives a prompt $P_k = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{y}_{k-1}, \mathbf{x}_k)$ containing a sequence of pairs of inputs and labels followed by a query input. Each input $\mathbf{x}_i \in \mathbb{R}^d$ is sampled from a distribution D_X and is labelled by a function $f : \mathbb{R}^d \to \mathbb{R}$ sampled from a distribution of functions $D_{\mathcal{F}}$. The goal of the model is to accurately predict $f(\mathbf{x}_k)$. Unlike works that study in-context learning on LLMs, in this framework, the model M is *trained from scratch* on various such prompts sampled according to distributions D_X and $D_{\mathcal{F}}$. Thus, this framework could be understood as a meta-learning approach, i.e., the models are trained to learn learning algorithms. Garg et al. [57] show that such trained Transformers perform well on a range of prediction and regression tasks on inputs in $\mathbb{R}^d$. Given the flexible and interpretable nature of the framework, several recent works [92, 3, 14] have adopted it to further understand in-context learning.

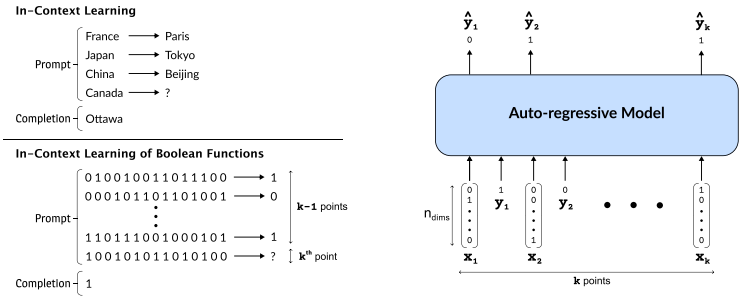


Figure 4.1: In-context learning of Boolean functions in our setup. In an autoregressive manner, given $k-1$ points of the form $(\mathbf{x}_i, \mathbf{y}_i)$ where $\mathbf{x}_i \in \{0,1\}^n$ and $\mathbf{y}_i \in \{0,1\}$, a model has to learn to accurately predict the label $\mathbf{y}_k$ for the k^{th} point.

Research Questions. Multiple works [14, 119, 162] have demonstrated that Transformers can learn various classes of real-valued functions in such in-context settings. This naturally motivates further questions: (a) What are the limits of the in-context learning ability of Transformers? (b) Is the attention mechanism essential for in-context learning? (c) Can Transformers exploit high-quality and informative examples to learn more efficiently? (d) To what extent do LLMs that are not specifically trained for these tasks carry the capacity to implement non-trivial learning algorithms on in-context examples?

In this chapter, we conduct an extensive empirical study with various classes of Boolean functions to make progress towards answering these questions. Since Boolean functions are widely studied in learning theory [111], the sample complexity and learnability of various function classes are well understood which helps us in designing the experimental framework. Additionally, unlike real-valued functions, the discrete nature of the inputs allows us to adopt the setting for LLMs and test their ability to implement learning algorithms. This chapter makes the following contributions.

In-context learning Boolean functions. Transformers trained from scratch perform well on a range of Boolean function classes, however, their performance does degrade on seemingly more 'complex' classes. In particular, when learning parities their performance is as bad as random guessing, but known algorithms can perform near-perfectly with the same size dataset. We also evaluate the performance of recently proposed alternatives to Transformers which were motivated by the desire to have sub-quadratic inference. Our results indicate that while these models match the Transformer's performance on most tasks, there remain some gaps in others.

Teaching Sequences. For a class of functions, a teaching sequence is a sequence of examples that can uniquely identify the function. Our results show that Transformers have the capacity to learn more sample-efficient algorithms when trained on such highly

informative sequences. For certain tasks, our results show that a single Transformer can learn two different algorithms and dynamically choose between them depending on the in-context sequence–it uses the sample-efficient version if the input has a teaching sequence, and falls back on the less sample-efficient algorithm otherwise.

Investigation with LLMs. The goal of recent work in stylized settings was primarily to understand whether LLMs can use in-context examples to learn novel tasks. However, it is still unclear if LLMs can implement learning algorithms rather than simply indexing from a list of tasks seen during pretraining. To understand whether LLMs can learn in a manner similar to models in the stylized setting, we investigate the performance of pretrained LLMs on the same tasks, by adapting them in two different ways. First, we use a frozen GPT-2 model, only training the input embedding layer, so as to be able to evaluate GPT-2 in the same stylized framework. Second, working with discrete inputs enables us to use the bit tokens directly to test the performance of LLMs by providing them in-context examples. Our results show that these models achieve non-trivial performance, competing with *nearest neighbour* classification. As we are sampling learning problems from a large combinatorial space, it is virtually guaranteed that these models are learning from in-context examples alone.

4.2 Setup for In-Context Learning

In-context learning. Our setup closely follows that of [57] and is similar to other recent works which study the ability of Transformers to learn to implement learning algorithms in their forward pass [119, 92, 162]. In this setup, a sequence model M (such as Transformers) is trained using N sequences, each sequence consisting of m labeled examples, $(\mathbf{x}_1, y_1, \ldots, \mathbf{x}_m, y_m)$. The model is trained for the next token-prediction task, except that we only care about every alternate token: if $\hat{y}_1, \ldots, \hat{y}_m$ are the m output labels predicted by the model, where $\hat{y_k} := M(P_k)$ is predicted using the prefix $P_k = (\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$, the loss on this sequence is given by $\frac{1}{m}\sum_{k=1}^{m} \ell(\hat{y}_k, y_k)$. To generate the set of N training sequences, we do the following: We sample m points $\mathbf{x}_1, \ldots, \mathbf{x}_m$, i.i.d from some distribution D_X over X, we sample a function $f \in \mathcal{F}$ from some distribution $D_{\mathcal{F}}$ over $\mathcal{F}$, and obtain the sequence $(\mathbf{x}_1, f(\boldsymbol{x}_1), \ldots, \mathbf{x}_m, f(\boldsymbol{x}_m))$. This is repeated N times, with a fresh sample of m points and a freshly sampled f. For an appropriate loss function ℓ, the empirical loss over N training examples is defined as,

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{m}\sum_{k=1}^{m}\ell(M(P_k^{(i)}), f_i(\mathbf{x}_k))\right). \tag{4.1}$$

We will primarily work with classes of Boolean functions $f : \{0,1\}^n \to \{0,1\}$, which take as input a vector of bits of fixed length and output either 0 or 1. We say that a model M can learn a class of functions $\mathcal{F}$ in-context if with high probability, for $f \sim D_{\mathcal{F}}$ and for a sufficiently large $k_{\mathcal{F}}$, for all $k \geq k_{\mathcal{F}}$,[1] $\Pr_{P_k \sim (D_X^k, D_{\mathcal{F}})}[M(P_k) \neq f(\mathbf{x}_k)] \leq \epsilon$. We would like to highlight the difference from the traditional statistical learning framework, where a single target function $f \in \mathcal{F}$ is learned from a random sample of labelled examples. In our setup, each sequence is a *learning problem* and the model is learning a learning algorithm.

Models. We primarily focus on Transformers [161] but we consider five different types of architectures for the majority of the experiments. These include a recurrent model (LSTM) [73], a state-space model (DSS) [67], a long convolutional model (Hyena) [127], and a hybrid model (RetNet) [149]. We consider decoder-only GPT-like architectures containing causal masking (see Figure 4.1 right). Note that each input $\mathbf{x}_k$ is a vector of dimension n. A linear map, $\{0,1\}^n \to \mathbb{R}^d$, with learnable parameters, is applied to each input and label vector to map it to a vector of the same dimension as the model width. The models receive as input $(\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{m-1}, \mathbf{y}_{m-1}, \mathbf{x}_m)$ sequentially and for each prefix P_k for $1 \leq k \leq m$, they predict the label $\mathbf{y}_k$ which is provided as the next token as in a language model.

Training. The models are trained from scratch to minimize the objective defined in Eq. 4.1 where cross-entropy is used as the loss function. The details of the hyperparameters and other aspects of implementation are provided in Section 4.6.4. We have made our source code available at https://github.com/satwik77/incontext-bool.

4.3 In-context Learning Boolean Functions

In this section, we investigate the abilities of autoregressive architectures to learn various classes of Boolean functions in-context. Our experiments are geared towards answering the following questions: (1) What classes of Boolean functions can Transformers learn in-context, and what are their limitations? (2) Is attention necessary

[1]The choice of $k_{\mathcal{F}}$ may depend on the function class, as for more complex classes, more examples may need to be seen before learning is possible.

for in-context learning, and what are the differences between the capabilities of Transformers and attention-free architectures?

4.3.1 Descriptions of Tasks

Boolean functions have been widely studied in the field of theoretical computer science [111]. Given that it is extensively studied in the field of learning theory, it is well-suited for our task, since the discrete nature of the domain allows us to also test actual LLMs apart from neural networks trained from scratch. Additionally, the learnability and the precise properties of each function class are well-understood, which aids in experimental design choices (such as the number of in-context examples necessary for learning a concept class) and in drawing inferences from the results.

Tasks 1-2 *Conjunctions and Disjunctions.* For the input domain $X_n = \{0,1\}^n$, any input $\mathbf{x} \in X_n$ denotes a possible assignment to n Boolean variables $x_1, \ldots, x_n$. A literal is either the variable x_i or its negation $\bar{x_i}$. A conjunction is simply an *and* ($\wedge$) of some subset of the $2n$ literals. For example, for $n = 10$, one possible conjunction could be $x_2 \wedge \bar{x}_6 \wedge x_7$ which outputs 1 only when x_2, x_7 are 1 and x_6 is 0. The class of Conjunctions contains all such conjunctions over $\{0,1\}^n$. The class of Disjunctions is defined similarly with the *or* ($\vee$) operation ($x_2 \vee \bar{x}_6 \vee x_7$).

Task 3 *Sparse Disjunctions* are a specific case of Disjunctions where the function over n variables depends on at most k variables where $k \ll n$. In other words, the function class contains all Disjunctions with at most k variables. In our experiments, we set $k = 3$. For functions over n variables, the total number of 3-sparse disjunctions is $\binom{n}{3}3^3$. Hence, in our experiments, we set a higher value for the n (n_dims= 50) parameter so that the number of functions in the class is at least a million.

Task 4-5 *DNFs and CNFs.* A Boolean function is in the class DNF if it is a disjunction of one or more conjunctions. Formally, for any $\mathbf{x} \in \{0,1\}^n$, a function f belongs to the class of DNFs, if it can be expressed in the form $f(\mathbf{x}) = (z_{1,1} \wedge z_{1,2} \wedge \ldots \wedge z_{1,k_1}) \vee \ldots \vee (z_{m,1} \wedge z_{m,2} \wedge \ldots \wedge z_{m,k_m})$ where $z_{i,j}$ are literals which are either the variables or their negation. Similarly, CNFs contain functions that are a conjunction of one or more disjunctions. While only the simplest of classification rules may be defined using conjunctions/disjunctions, a richer family becomes available when using DNF/CNF representations. Since DNFs/CNFs are conjectured to be computationally hard to learn, we consider 3-term DNFs and 3-clause CNFs, which are known to be efficiently learnable.

Task 6 *Sparse Majority.* A function in the class sparse majority is parameterized by a subset of variables and outputs the majority value of those variables. For example, if

the relevant variables are x_2, x_4, x_5 and x_7, then the output for any input $\mathbf{x} \in \{0,1\}^n$ will be 1 if more than 2 variables have value 1 and will be 0 otherwise. Since each function is uniquely identified by a subset of variables $S \subseteq [n]$, the total number of sparse majority functions is 2^n.

Task 7 *0-1 Threshold Functions* are functions of the form $\text{sign}(\sum_{i=1}^{n} w_i x_i - b)$ where $w_1, \ldots, w_n \in \{-1, 0, 1\}$ and $b \in \{-k, \ldots, k-1, k\}$. Instead of the usual $X_n = \{0,1\}^n$, the input space can be seen over $\{-1, +1\}^n$, which makes it easier to define the functions. In our experiments, we set $k = 3$ and $n = 28$. Note that Conjunctions and Disjunctions can be seen as special cases of Integer threshold functions where b is $\|w\|_1 - 1$ or -($\|w\|_1 - 1$). See that the total set of functions in the class of Integer threshold functions is $\Omega(3^n)$.

Task 8 *Integer Halfspace* contains functions of the form $\text{sign}(\sum_{i=1}^{n} w_i x_i - 0.5)$ where $w_1, \ldots, w_n \in \{-k, \ldots, k-1, k\}$ and $x \in \{-1, +1\}^n$. The total set of functions in the class is $\Omega(3^n)$.

Tasks 9-11 *Parity* is one of the most fundamental classes of Boolean functions; widely studied in learning theory and cryptography. A Parity function computes the XOR ($\oplus$) of some subset of the n variables. For example, a Parity function could be $x_2 \oplus x_4 \oplus x_5$. Each Parity function outputs 1 when an odd number of the variables are assigned the value 1 and outputs 0 otherwise. The class PARITY-n contains all 2^n possible Parity functions over $\{0,1\}^n$. We denote the class of sparse parities with PARITY-(n,k), which contain Parity functions with exactly k relevant variables (or bits) defined over $\{0,1\}^n$.

Task 12 The Nearest Neighbour task is intended to evaluate the ability of architectures to learn to implement the nearest neighbour algorithm. Unlike other tasks, it doesn't involve a class of functions. For this task, each example consists of 100 points $x_1, \ldots, x_100 \in \{0,1\}^n$ where the first k points are labelled 0 or 1 uniformly at random. The labels for the next $100-k$ points are determined by their nearest neighbour in the preceding input points. More specifically, for any point x_j where $j \in \{k+1, \ldots, 100\}$, let $p = \arg\max_{i \in \{1,\ldots,j-1\}} \frac{x_i^T x_j}{\|x_i\| \|x_j\|}$. Then the label y_j corresponding to x_j is y_p.

Size of Hypothesis set. Unlike regression problems considered in earlier works, the number of functions in these Boolean function classes is finite. However, see that, for inputs over $\{0,1\}^n$, the total number of possible Conjunctions (or Disjunctions) is 3^n. Hence, even for $n = 30$, the function class has over 100 trillion functions. Moreover, even for a prompt with $m = 50$ examples, the total number of permutations is 50! for one set of inputs and a single target function. This implies that the likelihood of encountering the same training example twice is extremely low and that it is (almost)

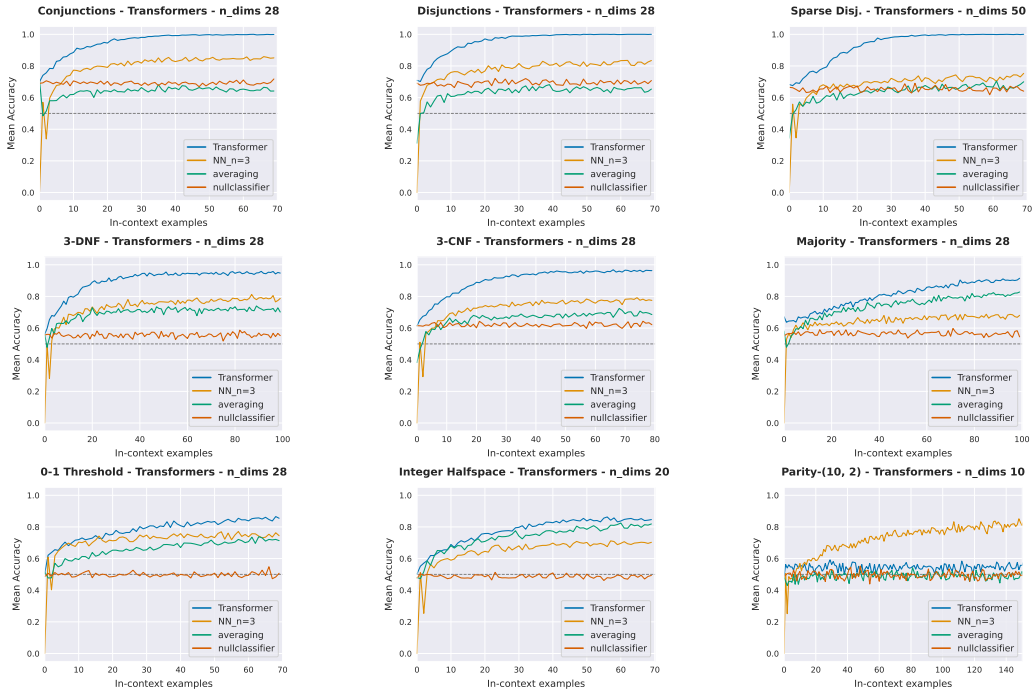


Figure 4.2: Performance of Transformers on in-context learning various classes of Boolean functions. The plots represent the mean accuracy of Transformers as a function of the number of in-context examples provided to the model. On certain classes of Boolean functions, such as Conjunctions and Disjunctions, Transformers perform in a near-optimal manner. On tasks such as 0-1 Threshold, Majority, and Integer Halfspaces, they perform better than baselines but do not achieve near-perfect accuracy even after observing 70 examples. On the other hand, on Parities, Transformers do not perform better than chance-level accuracy. Refer to Section 4.3 for details.

impossible for a practical-sized model to memorize all examples during training unless n and m are quite small.

4.3.2 Details of Input and Function Distribution

For the majority of the tasks, the inputs are sampled uniformly at random over $\{0,1\}^n$. For tasks such as Conjunctions, Disjunctions, and 3-CNF, a uniform distribution for the inputs over $\{0,1\}^n$ will lead to most examples being either positive or negatively labelled. In that case, even the Null classifier which always produces the same output, can achieve over 99% accuracy. While Transformers trained over uniformly distributed inputs for these tasks achieve near-perfect accuracy, it is, in some sense, trivial since they do not necessarily need to learn any algorithms and can achieve near-perfect accuracy by learning a constant function. Hence, for such tasks, we modify the input distribution such that at least 30% of the input points in the prompt have a positive

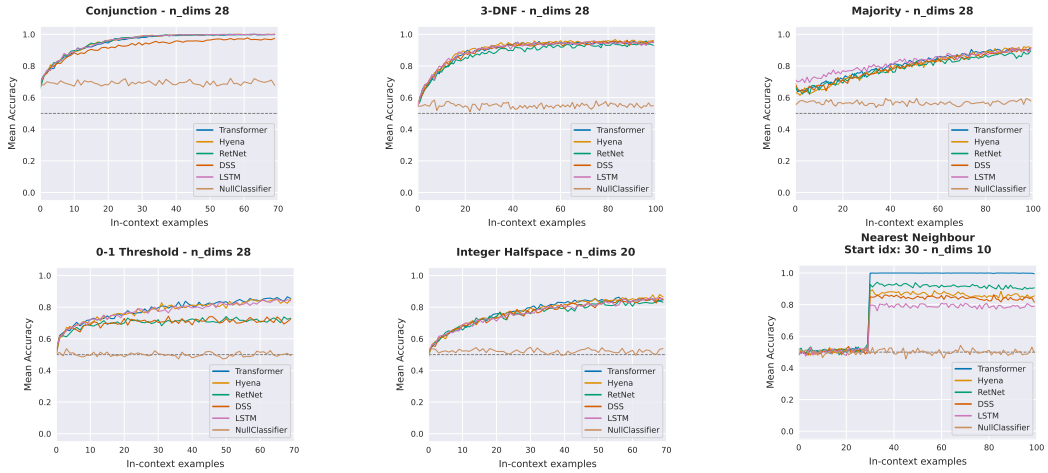


Figure 4.3: Performance of different architectures on 6 representative tasks.

(or negative) label to make it more well-balanced. In the modified distributions, we first sample k points for the prompt uniformly at random and then pick 30% of the input points. For these points, we change a subset of bits to 0 or 1 so that they will be labelled positively (for Conjunction) or negatively (for Disjunction) according to the target function. Note that this doesn't change the proportion of 0s and 1s in the modified inputs since the probability of a literal being positive or negative (complement) in the Conjunction is the same.

Distribution over Functions. For tasks such as Sparse Disjunctions and Parities, the functions are sampled uniformly from the set of all such functions. For both 0-1 Threshold functions and Integer halfspace, the w_is are sampled uniformly from their respective range, and for 0-1 Threshold functions, the b parameter is sampled uniformly from $\{-k, \ldots, k-1, k\}$. For tasks such as Conjunctions and Disjunctions, which are either AND ($\wedge$) or OR ($\vee$) of some of the $2n$ literals, the function is sampled such that each literal x_i or its complement $\bar{x}_i$ has a probability p of being in the sampled function. In other words, for each $i \in [n]$, the literal x_i has $p/2$ probability of being sampled, the literal $\bar{x}_i$ has $p/2$ probability and with $1-p$ probability the literal or its complement is not included. In our main experiments, we set $p = 30\%$ and also investigate the robustness of the trained model when evaluated on other distributions (other values of p) in Section 4.6.1. Similarly, for the Sparse Majority class, the probability of each literal being in the sampled function is p (there is no complement). For sampling each 3-DNF (or 3-CNF), we first sample three conjunctions (or disjunctions) with $p = 20\%$ and then take an AND (or OR) of those clauses.

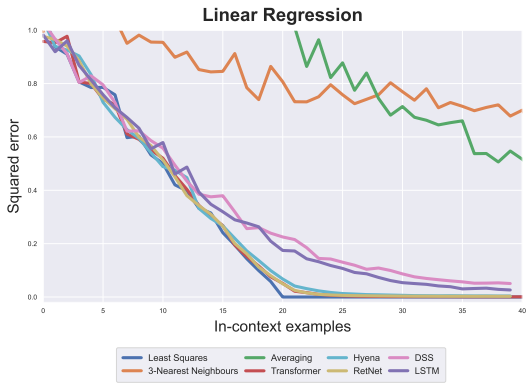


Figure 4.4: Performance of various architectures on the Linear Regression task from [57].

4.3.3 Details of Baselines

Below, we describe how each baseline model M predicts the label when provided with a prompt $P_k = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{y}_{k-1}, \mathbf{x}_k)$.

Null classifier. This classifier always produces the same output, i.e. $M(P_k) = 0$, irrespective of P_k.

Averaging estimator. This is an incremental learning classifier that captures the average relationship between input vectors and output labels for previously seen points. It predicts $M(P_k) = \hat{\mathbf{w}}^T \mathbf{x}_k$. The weights $\hat{\mathbf{w}}$ are computed as the average of the product of input vectors and their corresponding outputs over all previous points, i.e., $\hat{\mathbf{w}} = \frac{1}{k-1} \sum_{i=1}^{k-1} \mathbf{x}_i \mathbf{y}_i$.

Nearest neighbours (NN). This classifier aims to predict the output label based on the labels of the n most similar input vectors in the previously seen points. It predicts $M(P_k) = \frac{1}{n} \sum_{i \in S} \mathbf{y}_i$. Here, S is the set of indices of the n nearest neighbours of $\mathbf{x}_k$ among $\mathbf{x}_1$ to $\mathbf{x}_{k-1}$. For $k - 1 < n$, we average over all the $\mathbf{y}_i$s from 1 to $k - 1$, and for $k = 1$, we set $M(P_k) = 0$.

Feedforward Networks (FFN). We train a 1-hidden-layer ReLU FFN on the same number of examples as provided to sequence models in Table 4.1. We tune the model across width $\in \{100, 500, 1000\}$, optimizer $\in$ { 'adam', 'sgd'}, and learning rates $\in \{0.01, 0.005, 0.001, 0.0001\}$. We report the accuracy of the best model based on 1280 examples after being trained on $m - 1$ examples where m is given in Table 4.1 for each task.

4.3.4 Results

Table 4.1 summarizes the performance of various architectures on the Boolean function tasks considered in the work. The numbers indicate the accuracy of prediction for the last example in a prompt sequence, averaged over 1280 sequences. The accuracy curve of Transformers and baselines across all in-context examples is provided in Figure 4.2. The performances on Conjunctions and Disjunctions are almost identical in all the experiments we have conducted. Similarly, the performance on 3-CNF and 3-DNF, as well as among Parity tasks, is almost identical. The accuracy curves of all 5 architectures on representative tasks are provided in Figure 4.3.

Transformers show varied performance on Boolean functions. We find that on certain *simple* classes of functions such as Conjunctions and Disjunctions, Transformers achieve perfect accuracy within a relatively small number of examples. We observe that the performance of Transformers is close to that of feedforward networks (FFN) trained with gradient descent as well as known PAC-learning algorithms for the respective tasks (see Figure 4.19), in terms of sample complexity required to achieve perfect accuracy. On such tasks, we also find that they perform well on out-of-distribution inputs and functions; they also exhibit robustness to the presence of noisy labels during the training process (see Section 4.6.1 for more details).

While Transformers perform well on certain classes of Boolean functions, we find that their performance degrades across others. The tasks can be broadly divided into three categories. The first category contains tasks where Transformers can achieve near-perfect ($> 95\%$) accuracy. The second category contains examples where the model performs better than baselines but its accuracy saturates below 95%. The last category consists of Parities and Sparse Parities where Transformers and other models do not improve beyond chance-level accuracy even with a large number of training examples. While prior works [57, 119, 14] have observed that Transformers perform well on a wide range of tasks, our results highlight that there are certain classes of functions where they are imperfect (second category) and there are certain classes such as Parities where they seem incapable of finding any learning algorithm to solve the task. Note that, for Parity-$(10, 2)$, 140 examples are sufficient for feedforward networks trained with gradient-descent to achieve near-perfect test accuracy (see Figure 4.7). The tasks Parity-20 and Parity-$(20, 3)$ could be PAC-learned within 140 examples if the models could learn to apply the Gaussian elimination algorithm [56]. Hence, the failure of Transformers on the tasks in learning to learn Parities does not arise due to the lack of a sufficient number of in-context examples. We discuss and further explore the difficulty of learning Parities in Section 4.3.6.

Table 4.1: The performance of different architectures on in-context learning Boolean functions. The numbers depict the accuracy of a model while predicting the last input in the prompt. For instance, for prompts with m points, the number indicates the accuracy on the m-th input point based on the first $m-1$ points and labels. The descriptions of the baselines are provided in Section 4.3.3. For all values reported in the table, the standard deviation is within 0.5.

TASK	N_DIMS	POINTS	NULL	AVERAGING	NN	FFN	DSS	LSTM	RETNET	HYENA	TRANSFORMER
Conjunctions	28	70	69.9	68.2	81.3	94.0	97.2	100.0	99.4	100.0	100.0
Disjunctions	28	70	69.8	67.5	80.3	93.4	97.5	100.0	99.7	100.0	100.0
Sparse Disjunctions	50	70	63.9	70.4	71.7	91.2	99.5	100.0	98.2	100.0	99.9
3c-CNF	28	100	59.3	66.6	79.6	91.5	93.3	94.3	91.5	95.5	95.1
3t-DNF	28	100	55.6	70.3	80.4	90.9	94.7	95.5	92.9	96.0	95.2
Majority	28	100	55.5	80.9	67.8	84.0	90.7	91.2	89.6	91.5	91.5
0-1 Threshold	28	70	50.5	70.7	77.1	89.6	72.4	83.9	72.9	85.0	85.3
Integer-Halfspace	20	70	51.4	81.7	70.0	90.6	84.0	84.6	83.1	86.9	86.4
Parity-(10, 2)	10	140	50.6	48.5	81.7	100.0	49.7	51.1	48.7	52.0	53.7
Parity-(20, 3)	20	140	50.0	50.7	55.3	50.2	51.2	50.0	51.1	47.6	49.1
Parity-20	20	140	49.2	50.6	49.2	50.6	47.8	49.6	50.3	51.1	50.6
Nearest Neighbor	10	100	49.6	66.6	100.0	N/A	81.5	79.1	90.6	85.5	100.0

Transformers vs. other Architectures. For the majority of the tasks considered in this work, we observe that attention-free architectures perform almost as well as Transformers. The efficacy of recurrent and long-convolution models on multiple tasks such as Conjunctions and Disjunctions indicates that the in-context learning phenomenon in the sense of implementing learning algorithms is not particular to Transformers. At the same time, we find that certain differences between Transformers and their recently proposed alternatives evidently exist. On tasks such as 0-1 Threshold functions, we find that RetNet and DSS perform worse than Transformers (see Table 4.1). With further experiments on linear regression tasks proposed in previous works, we find that recurrent models such as DSS and LSTMs struggle to match the performance of Transformers (see Figure 4.4). The closest to Transformers in terms of performance is Hyena, which matches Transformer's performance on all tasks apart from the nearest-neighbor (NN) task. The NN task tests the ability of models to implement the nearest-neighbor algorithm as described in Section 4.3.1. Our results indicate that while recently proposed architectures with sub-quadratic inference time can perform in-context learning, there still exists a minor performance gap between these architectures and Transformers.

4.3.5 How do Transformers Learn Conjunctions?

In this line of work, each sequence of examples or prompt $(\mathbf{x}_1, f(\boldsymbol{x}_1), \ldots, \mathbf{x}_m, f(\boldsymbol{x}_m))$ can be seen as a learning problem. In the traditional statistical learning framework, while learning from examples labelled by a particular function $f \in \mathcal{F}$ (Conjunction), the goal of a learning algorithm is to find the target hypothesis function $f \in \mathcal{F}$. In this case, during the meta-learning stage, we are optimizing a Transformer (or other networks) to find a learning algorithm rather than a target function. Hence, in that sense, the target function in this meta-learning-like setup is the optimal learning algorithm for the class of functions. It is natural to wonder how Transformers fare against known optimal learning algorithms for the problems considered in this work. While the optimal learning algorithms for all classes considered here may not be known, there are some known algorithms for learning classes such as Conjunctions and Disjunctions, which are in some sense near-optimal.

Classes such as Conjunctions and Disjunctions are known to be efficiently PAC-learnable with $O(n)$ examples. The classical algorithm (see Algorithm 1) for learning Conjunctions works as follows, it starts with a hypothesis h which has all $2n$ literals $(h = z_1 \wedge \bar{z}_1 \wedge z_2 \wedge \bar{z}_2 \wedge \cdots \wedge z_n \wedge \bar{z}_n)$. It goes through each example and only checks the positive examples. For each positive example, it finds the literal which are not satisfied and drops those literals from the hypothesis h. This is because we know that if a literal is not satisfied and the label of the input is 1 then the target conjunction does not have that literal. For any set of examples, this algorithm returns a conjunction consistent with all examples and PAC-learns the class Conjunctions. A similar algorithm can be derived for Disjunctions as well. See [160] for more details.

4.3.5.1 Results: Transformers as Learning Algorithms

Performance Comparison. Figure 4.19 compares the performance of the classical algorithm for Conjunctions (Algorithm 1) and Disjunctions with the performance of Transformers. There are a few things to note here: (a) These classical algorithms are near-optimal and are guaranteed to work for any distributions over the input. They are optimal in the sense that the lower bound on the sample complexity of learning Conjunctions/Disjunctions is $O(n)$ and these PAC-learn these classes with $O(n)$ samples. (b) Transformers can perform better on the first few examples because they have knowledge about the distributions of inputs and functions since they are exposed to them in the meta-learning stage. So in that sense, they are closer to the Bayes-optimal-estimator [119] than the classical algorithm in their behaviour. However,

Algorithm 1 CONJUNCTIONS Classical Algorithm

Input: n, $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$
Initialize: $h = z_1 \wedge \bar{z}_1 \wedge z_2 \wedge \bar{z}_2 \wedge \cdots \wedge z_n \wedge \bar{z}_n$
for $i = 1, \ldots, m$ **do**
    Pick $(\mathbf{x}_i, y_i)$ from S
    if $y_i = 1$ **then** ▷ Ignore negative examples
        for $j = 1, \ldots, n$ **do**
            if $x_{i,j} = 0$ **then**
                Drop z_j from h ▷ j^{th} bit of i^{th} instance is 0
            else
                Drop $\bar{z}_j$ from h ▷ j^{th} bit of i^{th} instance is 1
            end if
        end for
    end if
end for
Output: h

unlike the provable PAC-learning algorithms, the algorithms learned by Transformers are not guaranteed to work for arbitrary distributions. (c) Both Transformers and the classical algorithm reach (near) perfect accuracy after observing almost the same number of $O(n)$ examples.

Interpretability. We explore how Transformers could be (in-context) learning Conjunctions in this section. To simplify things, we will focus on Monotone-Conjunctions, which contain only positive literals and no negative literals. That is, a function such as $f = z_1 \wedge z_3$ is in the class of Monotone-Conjunctions but $\bar{z}_1 \wedge z_3$ is not since $\bar{z}_1$ is a negative literal.

Approach. The key observation behind our approach to probing Transformers is the observation that Monotone Conjunctions can be exactly learned by membership queries alone. In other words, if there is a monotone-conjunction f over $\{0,1\}^n$ which is unknown but we can query the label $f(\mathbf{x})$ for any input $\mathbf{x}$, then we can exactly identify the Conjunction by querying n examples.

The n examples $\mathbf{e}_1, \ldots, \mathbf{e}_n$ will be such that the example $\mathbf{e}_i$ has value 0 at the ith coordinate and value 1 everywhere else. For instance, over 4 dimensions, the vectors $\mathbf{e}_1 = [0111], \mathbf{e}_2 = [1011]$ and so on. See that for any unknown conjunction f over $\{0,1\}^n$, if we query the function for input $\mathbf{e}_i$ and the label $f(\mathbf{e}_i) = 0$, then it implies that the literal z_i is in the conjunction f. Hence, by querying all n such examples $\mathbf{e}_1, \ldots, \mathbf{e}_n$, we can identify any unknown monotone conjunction.

Interpreting Transformers. We train the Transformers for the class of monotone conjunctions by sampling prompts and optimizing the model similar to previous

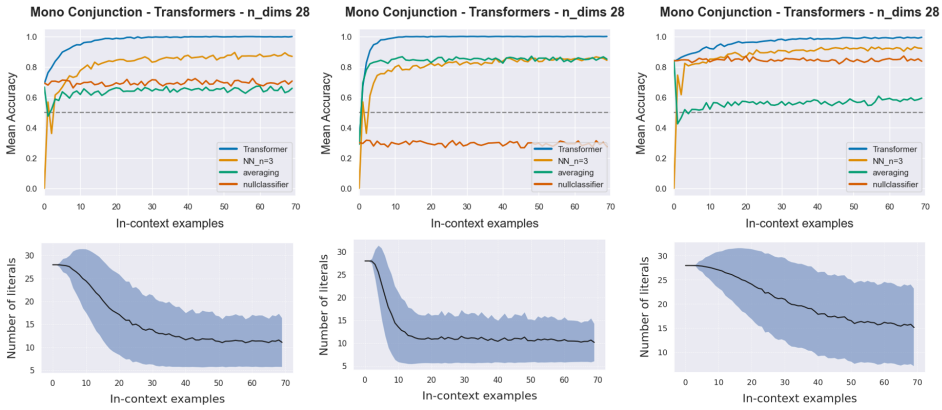


Figure 4.5: Extracting the target (Mono) Conjunction from Transformer during in-context learning. The upper row depicts Transformer's performance across in-context examples and the lower row shows the number of literals in the conjunction extracted from the Transformer at that point. The values are averaged across 1k prompts. The three columns are for three different distributions over inputs (at test time). First column: $\approx 30\%$ of inputs in the prompt are positive, second: $\approx 70\%$ of inputs are positive, third: $\approx 15\%$ of inputs are positive.

experiments (as described in Section 4.2). We are interested in figuring out what kind of algorithm it applies to learn conjunctions in-context. During evaluation, the model M sequentially receives prompts of the form $P_k = (\mathbf{x}_1, y_1, \ldots, \mathbf{x}_{k-1}, y_{k-1}, \mathbf{x}_k)$ for $1 \leq k \leq m$. After receiving each prefix P_k, we evaluate the model M on the n examples $\mathbf{e}_1, \ldots, \mathbf{e}_n$ to identify the function used by the Transformer model M. More specifically, let $P_k^{(i)} = (\mathbf{x}_1, y_1, \ldots, \mathbf{x}_k, y_k, \mathbf{e}_i)$, then for each $k \in [0, \ldots, m]$, we evaluate the model on $P_k^{(1)}, \ldots, P_k^{(n)}$.

See that if the Transformer model M makes its prediction based on some unknown Conjunction, then we can recover it during each step of in-context learning. However, it is not necessary that the model M predicts according to a conjunction and may be using some other form of function, in which case this approach is not guaranteed to work. Note that, since Transformers reach perfect accuracy after observing a certain number of examples (Figure 4.2 top-left) and also achieve that on out-of-distribution examples, it is reasonable to expect it to predict according to some underlying conjunction as it sees more examples. Nonetheless, we apply the approach described above to extract the target Conjunction at each step of the in-context learning process.

Learning Mono Conjunctions. For Monotone-Conjunctions, the PAC-learning algorithm is a simpler version of Alg. 1. It starts with a conjunction with all literals $h = z_1 \wedge z_2 \cdots \wedge z_n$ and while going through m examples, it only looks at the positive examples. For any positively labelled example x_i, if any coordinate $x_{i,j} = 0$, then it

drops the literal z_j from the hypothesis h and outputs h after going through all m examples.

Results. We find that Transformers behave in a manner very similar to the algorithm described above but not in an exact way. We evaluate the model on 1k prompts containing 70 examples each and extract the literals at each step, beginning from no examples to all 70 examples. We observe that in the beginning, the target function has all literals similar to the classical algorithm. Moreover, as the model observes more examples, the number of literals decreases monotonically and finally converges to the correct target conjunction, which was used to label the examples in the prompt. However, unlike the classical algorithm, the Transformer model does not drop literals right after seeing a positive example; we find that it drops a larger number of literals together after observing a series of positive examples.

Figure 4.5 depicts the number of literals in the extracted conjunction across in-context examples for three different input distributions. The model was trained according to the first distribution, where approximately 30% of the examples in the prompt are positively labelled and then tested on three different distributions. Similar to the classical algorithm, the model seems to start with all literals and then gradually drops literals as it sees more examples. Upon manual inspection, we found that the model does not always drop literals right when it sees a positive example. At the same time, as can be seen from Figure 4.5, when the prompt contains more positive examples (second column), the model drops literals more quickly and reaches perfect accuracy with fewer examples overall. On the other hand, with fewer positive examples, the Transformer seemingly drops literals later (third column in Figure 4.5) and needs more examples to reach near-perfect accuracy. This behaviour is similar to the classical algorithm. Additionally, we compute the number of times the Transformer model drops an incorrect literal (a literal that is present in the target conjunction) at any point in time during the in-context learning phase. We find that across all three input distributions, it drops a literal incorrectly in less than 0.1% out of 1k prompts containing 70 examples each. In over 95% of the prompts, it converges to the correct target conjunction. Our results suggest that Transformers implement an algorithm akin to the classical one, initially starting with all literals and then progressively dropping them as they encounter more and more positive examples. However, deviations from the classical algorithms exist that remain to be understood.

4.3.6 Curious Case of Learning Parities

An interesting and somewhat surprising phenomenon is the failure of Transformers and other architectures on the class of Parities and sparse Parities. In contrast to other function classes where models are able to learn some learning algorithm, for Parities, they do not seem to be able to find any kind of learning algorithm that performs beyond chance-level accuracy.

We examine the gradients and the model parameters during the training process. In Figure 4.6, we show the mean of gradients for all parameters of attention and feedforward networks in the Transformer. Observe that the gradients are very close to 0 after a few iterations of training. Note that this is in contrast to what we observe while training Transformers for tasks such as Conjunctions, where gradients are not so close to 0. However, the exact reason why Transformers fail to learn anything is unclear and is left as an open problem.

The problem of learning Parities and Sparse Parities has well-known hardness results [83] in the Statistical Query framework, indicating that Parities require $2^{\Omega(n)}$ queries and Sparse Parities require $n^{\Omega(k)}$ queries (where k is the number of relevant bits). Gradient-based methods are considered to be only as powerful as SQ-learning algorithms in finite-precision settings, and hence any model, such as Transformers, applying a gradient-based algorithm to learn in-context would require an exponential number of steps to predict accurately. For sparse parities PARITY-$(10, 2)$, we provide more than n^k examples in the prompt and hence a model applying gradient-based learning, such as FFNs, can solve it (see Fig. 4.7 Left). For instance, feedforward networks trained with gradient descent on the same number of examples as provided to Transformers for ICL are able to achieve near-perfect accuracy. However, Transformers and other architectures do not seem to make any progress during their training process. Also, note that the hardness of learning Parities primarily holds for uniform distribution over inputs and does not apply to input sets with teaching sequences that are no longer uniform.

Parities are PAC-learnable using the Gaussian elimination algorithm [56] with $O(n)$ examples, and if any of the architectures could learn to implement the algorithm, then they would have been able to solve it with $O(n)$ examples.

Transformers vs LSTMs. An interesting difference we observed is that for the PARITY-$(10, 2)$ problem, LSTMs achieve 100% accuracy if the test examples are labelled by the same set of functions as the training examples. This is not true for Transformers. For the PARITY-$(10, 2)$ problem, there are only 45 different functions. For the results in Table 4.1, we ensure that the training examples are labelled by 23

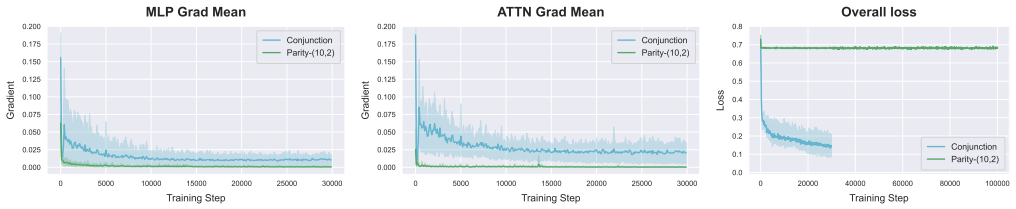


Figure 4.6: *Left:* Figure depicting the mean of the gradients of parameters such as MLP and attention parameters across different iterations of training on Parity-(10, 2) and Conjunctions. *Right:* The loss of Transformers on Conjunctions vs Parity-(10, 2) across different iterations of training.

of those functions and the test examples are labelled by the remaining and unseen functions. However, if we uniformly sample a function out of the 45 functions during both training and evaluation, then we observe that LSTMs achieve perfect accuracy during evaluation. This is interesting since the sequence of input points $x_1, \ldots, x_k$ are new during evaluation and hence is not a clear case of memorization. On the other hand, since they do not generalize to unseen functions, it is difficult to claim that LSTMs can learn sparse parities in-context.

Curriculum learning. For Sparse Parities, we also tried curriculum learning similar to [57] but did not observe any improvement. In particular, we trained Transformers for the problem of PARITY-$(14, 2)$. In the curriculum learning setup, the input dimension is initially smaller, i.e., 9, and it increases incrementally by 1 after every 50k steps. Hence, effectively for the first 50k steps, the model is trained for PARITY-$(9, 2)$, then PARITY-$(10, 2)$ and so on eventually ending at 14 (trained 250k steps after reaching 14). The model is trained for 500k steps in total. The number of examples in the prompt m is 120 in the beginning and increases by 20 every 50k steps and eventually stays at 200. We start at input dimension 9 because below that the total number of Boolean inputs is very small and the prompt will cover over half of all possible inputs. We do not find any improvement with the curriculum learning setup as well. The loss curve is identical to the one in Figure 4.6 (right) and is hence omitted.

4.4 In-context Learning with Teaching Sequences

Premise. Our experiments in the previous section as well as previous works investigate the abilities of models to perform statistical learning from arbitrary distributions in an in-context setting. Given that we are exploring the ability of sequence models to learn 'learning algorithms', one could wonder: *Can sequence models find learning algorithms*

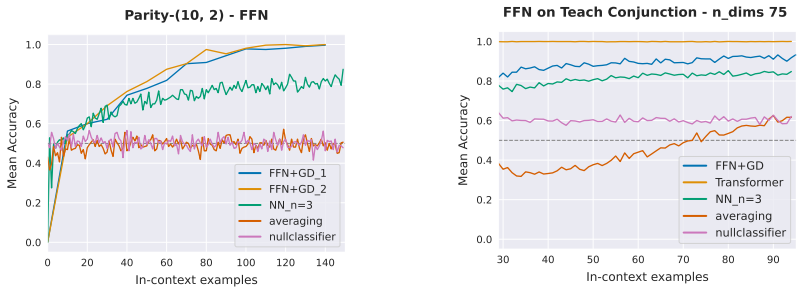


Figure 4.7: Left: Performance of Feedforward networks (FFN) trained with gradient descent on Parity-(10, 2) with the same number of examples as provided to models in ICL setting. *Right:* Performance of FFN on Teach Conjunction task.

which are more sample-efficient when they are provided with a more informative sequence of examples? We explore this question in this section.

4.4.1 Preliminaries on Teaching Sequences

Teaching Sequences. The concept of teaching sequences and teaching dimensions was introduced in [64]. For any class of functions $\mathcal{F}$, the teaching sequence of a function f in $\mathcal{F}$ is a sequence of labeled instances $(\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_m, \mathbf{y}_m)$ such that it unambiguously identifies f in $\mathcal{F}$ – only the function f is consistent with the sequence of labeled instances. Let $\mathrm{T}(f)$ be the set of all teaching sequences for the function f. The teaching dimension [64] of a function class is defined as, $\mathrm{TD}(\mathcal{F}) = \max_{f \in \mathcal{F}}(\min_{\tau \in \mathrm{T}(f)} |\tau|)$.

Conjunctions and Disjunctions. The teaching sequence for Conjunctions can be created as follows. Suppose f is a conjunction of k literals. We will create $k+2$ points. For the first two points, the variables corresponding to all literals in the Conjunction are set to True, that is, they are 1 for positive literals in the Conjunction and 0 for negative literals. In the first point, all variables that do not affect the output of the function are set to 0, and in the second point, all of them are set to 1. The next k points will correspond to each of the literals in f. For each literal z_i, we create a point $x \in \{0,1\}^n$ where the variable of x at i is 0 if z_i is in the Conjunction and is 1 if $\bar{z}_i$ is in the Conjunction. The variable corresponding to every other literal in f is set to 1 if it is a positive literal and is 0 if it is a negative literal. The variables which do not affect the output of the function are all set to 0. Essentially, these examples show that even if the variables corresponding to every other literal in the function are set to true, the function value will still be 0 because one of the literal values is set to false. The teaching sequences for Disjunctions are created in a similar manner.

See that since for every Conjunction or Disjunction, the length of the teaching sequence is $k+2$, the teaching dimension is $n+2$, and for each function with k literals, the

exact function can be identified with $k+2$ points in the prompt example or a training set. Note that the sample complexity of $O(n)$ for Conjunctions or Disjunctions is with respect to PAC-learning in the sense that with $O(n)$ points, one could guarantee that the error will be within some ϵ for arbitrary distributions. However, the exact number of samples will depend on the value of ϵ and could be much larger than k or n when the error ϵ is required to be very small.

DNFs and CNFs. For simplicity, let's consider Monotone 3-DNFs which only have positive literals. For each clause or Conjunction in the DNF, we create $k+1$ points. The first point is such that variables corresponding to every literal in the clause are set to 1 and all other variables are 0. The other k points are created as follows. For every literal in the clause, there is one point where the variable corresponding to the literal is 0 and the variables corresponding to other literals in the clause are set to 1. The variables that are not in the clause are set to 0. For instance if the input is over $\{0,1\}^4$ and the clause is $x_1 \wedge x_3$, then the three points corresponding to the clause will be $[1010], [0010]$, and $[1000]$. The teaching sequence for the 3-DNF is the union of the teaching sequence of its three clauses and hence will have size $l+3$ where l is the total number of literals in all clauses. The teaching sequence for 3-CNF can be created in a similar manner.

Sparse Parities. For sparse parities with k relevant bits, the teaching sequence has k points where in each point the variable corresponding to one of the relevant bits is set to 1 and every other bit in the input is set to 0. Note that, this is a teaching sequence of the class PARITY-(n,k) which have exactly k relevant bits. This would not work as a teaching sequence for the entire set of Parity functions PARITY-n.

For further details on teaching sequences, see [64].

4.4.2 Experiments with Teaching Sequences

Experimental Setup. Every aspect of the experimental setup except the data distribution remains identical to the previous experiments. For each sequence during training and evaluation, the first t points are a teaching sequence of the target function and the next $m-t$ points are sampled from the distribution D_X. The sampling of functions, the training and the evaluation process remain the same. Although we primarily focus on Transformers, most results in this section hold for all architectures.

Tasks. We experiment with 5 tasks: Conjunctions, Disjunctions, CNFs and DNFs, and sparse parities. Every conjunction (or disjunction) has a teaching sequence of length $k+2$ where k is the number of literals in the function. Every 3-term DNF or CNF has a teaching sequence of length $l+3$ where l is the total number of literals in

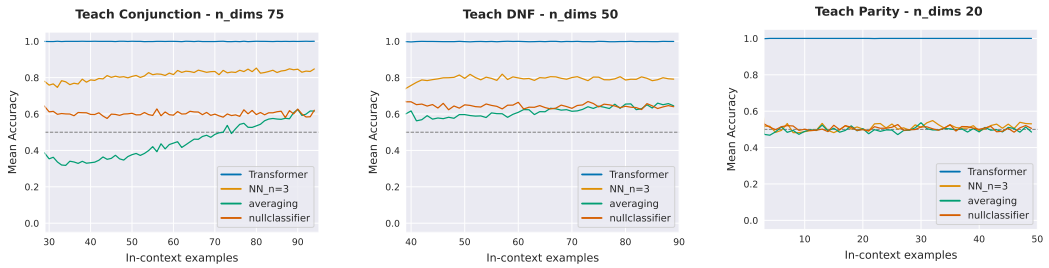


Figure 4.8: Performance of Transformers on various tasks with Teaching Sequences. The plots depict the performance of models right after the teaching sequence. Refer to Section 4.4 for more details.

all 3 terms. The teaching sequence of each sparse parity function is of length k where k is the number of relevant bits upon which the Parity function is computed. For each task (such as Conjunctions), we refer to the version with teaching sequence as Teach <task-name> (e.g. Teach Conjunction).

Transformers succeed in learning from Teaching Sequences. For all 5 tasks, we find that Transformers achieve perfect accuracy on subsequent points after receiving the teaching sequence. Figure 4.8 shows the accuracy curve of Transformers and baselines on three representative tasks after receiving the first t points containing the teaching sequence. Note that, the accuracy of Transformers stays (close to) 100% after receiving the teaching sequence. It is interesting to see that Transformers succeed in learning with teaching sequences for tasks like DNFs and sparse parities where they struggled to learn in the vanilla setting.

By definition, the teaching sequence is the smallest sequence required to learn the target function. Hence, it can often be much smaller than the sample complexity for a learning algorithm to learn from arbitrary distributions such as the ones considered in the previous section. Thus, our experiments show that models predict perfectly with much fewer examples when provided with these teaching sequences. This is not true for FFNs trained with gradient descent which fail to predict accurately given only the teaching sequence during training (see Figure 4.7 right). FFNs+GD is a more general-purpose algorithm and is hence not the most optimal for specific problems.

Two distinct algorithms to learn one task. An interesting finding is that Transformers seem to learn two different algorithms to learn Conjunctions depending on the data distribution D_X during the training process. See that (Figure 4.9 left) when Transformers are trained to perform in-context learning with Conjunctions in the vanilla setting (as in Section 4.3) without teaching sequences, and tested on examples (or Prompts) containing teaching sequences, they are not able to leverage the teaching

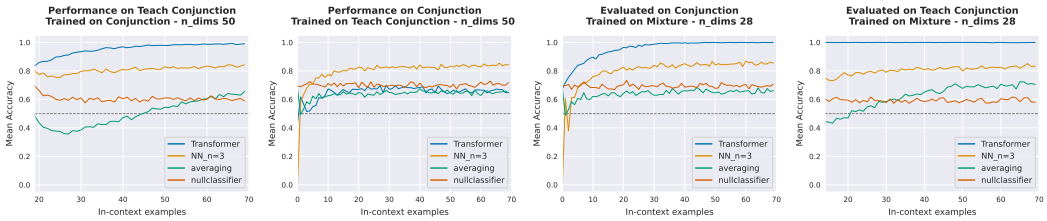


Figure 4.9: *Left*: depicts the performance of Transformers trained on examples with the vanilla distribution and tested on in-context examples with Teaching Sequences. *Center-left*: Transformers trained with teaching sequences and tested on in-context examples without teaching sequences. *Center-right* and *right*: depict the performance of Transformers trained on a mixture of Conjunction and Teach Conjunction and tested on the individual tasks. See Section 4.4 for more details.

sequence and require more examples to predict accurately. The algorithm learned with standard Conjunctions still works on examples with teaching sequences even if it is not as sample-efficient. On the other hand, when Transformers are trained with teaching sequences and are tested in the standard setting without teaching sequences, they do not perform well since the learning algorithm relies on using the first t examples containing the teaching sequence (Figure 4.9 center-left). These results indicate that models can learn two distinct algorithms depending on the distribution of inputs provided during training.

We conducted another experiment where we trained a Transformer on a mixture of examples with and without teaching sequence. During training, we sample an example (or Prompt sequence) with a teaching sequence with probability $\frac{1}{2}$, and without a teaching sequence with equal probability. We evaluate the model on both tasks separately, one that contains examples with teaching sequences and one without them. We find that the same Transformer model could achieve near-optimal performance for both tasks – when provided with teaching sequences, it behaves like the model that is trained on just the Teach Conjunction task (Figure 4.9 Right) and when provided with examples without teaching sequences, it behaves like a model trained on just the Conjunction task (Figure 4.9 center-right). This indicates that Transformers can learn two distinct learning algorithms for Conjunctions and implement the optimal one depending on the sequence of in-context examples. This highlights the versatility of neural sequence models in being able to find separate learning algorithms with respect to the data distribution for solving the same task.

4.5 Investigations with Pretrained Models

Premise. While several works have investigated how well Transformers can be trained to learn various kinds of functions in-context, it is unclear how these findings relate to LLMs, which was the primary motivation behind this line of work. This raises a very natural and intriguing question: *Can pretrained language models predict accurately by implementing any kind of learning algorithm when provided with prompts of the form* $(\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_k)$*?*

While prior works focused on learning real-valued function classes, evaluating LLMs on those tasks is not straightforward since LLMs receive and produce discrete values. Our setup involves discrete Boolean functions, which enables us to analyse the performance of LLMs in applying learning algorithms for solving these tasks. Additionally, as we are picking the target function randomly from an exponentially large set, we can be essentially sure that the learning is happening from in-context examples only, allowing us to remove confounding factors such as memorisation during pre-training.

Setup. Recall that, in our experiments in the previous sections, when a model is provided with a sequence of input-output pairs $(\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_m, \mathbf{y}_m)$, each input $\mathbf{x}_i$ is a single token. Additionally, LLMs do not have embeddings for each of the Boolean inputs $\{0,1\}^n$ as opposed to the linear map $\mathbf{W} : \{0,1\}^n \to \mathbb{R}^{\text{width}}$ learned by the models in the ICL setup of previous experiments. To address that, we experiment with pretrained models in two different settings, one which is closer to the ICL setup in previous experiments and another which is very close to the practical usage of the LLMs.

(a) Frozen GPT. In the first setting, we take a GPT-2 model and add learnable input and output layers at the beginning and end of the Transformer architecture. The weights of the GPT-2 model are frozen while the embedding matrix is replaced by a learnable function $I : \{0,1\}^n \to \mathbb{R}^{\text{width}}$ which is an FFN with one hidden layer. Similarly, the output layer is replaced by an FFN $F : \mathbb{R}^{\text{width}} \to \{0,1\}$. During experiments, we train the input and output layers to minimise the objective in Eq. 4.1 while keeping the weights of the pre-trained GPT-2 frozen. We compare it with the performance of a randomly initialised Transformer model (same architecture as GPT-2), undergoing the same training process with learnable input-output mappings and frozen Transformer weights. The goal of this setup is to understand whether large-scale text pre-training imparts some ability to the Transformer model that enables it to implement learning algorithms in-context.

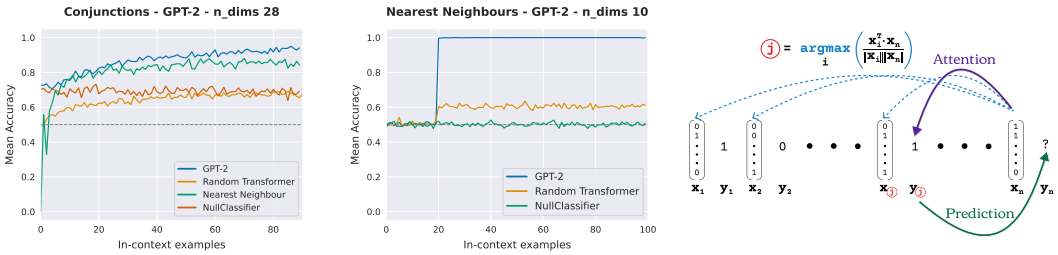


Figure 4.10: Experiments with frozen GPT: *left*: Performance of frozen GPT-2 on the Conjunction task, *center*: Frozen GPT-2 model learns to perfectly implement the nearest neighbours algorithm, *right*: Illustration of attention patterns in GPT-2 while solving the Nearest Neighbour task.

(b) Direct Evaluation. We directly evaluate several LLMs on the task of learning Boolean functions. In this setup, each input example $\mathbf{x}_i \in \{0,1\}^n$ is provided as a sequence of tokens $x_1, \ldots, x_n$ where $x_i \in \{0,1\}$. None of the model's parameters are modified and the original embedding matrices are used to represent the tokens 0 and 1. We evaluate the models by providing them with a simple prompt containing two parts: the first part is a simple instruction asking the model to act as a learning algorithm and predict the label corresponding to the final input, and the second part consists of a series of input sequences and their corresponding labels followed by the test input sequence (see Figure 4.11). We evaluate the predictions of the model at prompts of different lengths (i.e., different numbers of in-context examples). To estimate the accuracy for a particular number of in-context examples, we repeat the experiment 100 times (i.e., with 100 different functions of the same class). We experiment with state-of-the-art LLMs GPT-4 [114], GPT-3.5-Turbo [28, 117] as well as the open-source model LLaMA-2-70B [157].

4.5.1 Frozen GPT Experiments

Details of the Setup. With the experiments with frozen pretrained models, we aim to explore if pretrained models learn any mechanism that is useful for solving tasks in the meta-learning-like stylised setup.

The experiments follow the same framework as described in Section 4.2 with the exception that the weights of the Transformers are not modified or updated during the training process apart from the input embedding layer and the final output layer. We use the GPT-2 XL (1.5B parameters) model from Hugging Face [171].

In the stylised setup, since each Boolean input $x \in \{0,1\}^n$ is provided as a single token, the original embeddings and tokenizer of the pretrained model cannot be used for prediction. Hence, we replace the original embedding layer with a 1-layer ReLU

FFN: $\{0,1\}^n \to \mathbb{R}^{\text{width}}$ which produces an embedding for each $x \in \{0,1\}^n$. Similarly, we add an output FFN that predicts the final output given the vector produced by the pretrained model.

Similar to other experiments, we train the model on Conjunctions and the nearest neighbour task by sampling functions and input points to create training examples. The training process is used to update the embedding and output layer, and the weights of the Transformers are fixed.

Results with Frozen GPT. We evaluate the performance of a frozen GPT-2 model on tasks such as Conjunctions and Disjunctions. Figure 4.10 depicts the performance on Conjunctions (the behaviour on Disjunctions is almost identical). We find that the GPT-2 model performs relatively better than the nearest neighbour baseline and much better than a randomly initialized model on these tasks. However, it still does not come close to achieving the near-perfect accuracy obtained by fully trainable Transformer networks as in Section 4.3.

GPT-2 can implement Nearest Neighbour. Given the observation that the performance of GPT-2 was close to the nearest neighbour algorithm, we examined if a frozen GPT-2 model can implement the nearest neighbour (NN) algorithm. We designed the NN task to test this hypothesis. In this task, each prompt contains 100 points where the first 20 points are labelled 0 or 1 uniformly at random and the subsequent 80 points are labelled according to the nearest neighbour algorithm. We then evaluate the GPT-2 model in the frozen-GPT setup described earlier and find that it can achieve near-perfect accuracy on the 80 points labelled with the nearest-neighbour algorithm (see Figure 4.10 center). Moreover, upon analysing the attention heads we found heads which closely implemented nearest neighbours — for an input $\mathbf{x}_i$, the attention head attended over $\mathbf{y}_j$ where $\mathbf{x}_j$ is the nearest neighbour of $\mathbf{x}_i$ among $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$ (illustrated in Figure 4.10 right). These heads are reminiscent of induction heads [112] where, instead of matching prefixes, they can find the nearest neighbour over the input space. Since the weights of the Transformers are frozen and only the input embedding and output layers are updated during the training process, the mechanisms to solve tasks such as NN and Conjunctions must have been learned during pretraining on language data. Moreover, a Transformer of the same size with randomly initialized weights is unable to perform much better than chance-level accuracy.

Detecting NN heads. Our methodology for detecting attention heads that directly implement the nearest neighbours algorithm is similar to the one used by Olsson et al. [112] for finding induction heads. While training the model on the Nearest Neighbours

You are given some examples of inputs and their corresponding labels. You need to learn the underlying boolean function represented by these input-label examples. Predict the label (either 0 or 1) for the final input.

Input: 0 0 0 1 0
Label: 0
Input: 1 0 0 0 1
Label: 0
Input: 0 0 0 0 1
Label: 0

(... more exemplars ...)

Input: 1 1 1 0 1
Label: 1
Input: 1 1 1 0 0
Label: 0
Input: 0 1 1 0 0
Label:

Figure 4.11: An example of a prompt provided to the LLM in the direct evaluation experiments. Here, the model has to learn a function from the Majority class in 5 dimensions and predict the label for the last input. The prompt consists of a natural language instruction and k example points of the function.

task, for each attention head in the model, we compute a 'nearest-neighbours score (nn-score)'. After every 500 training steps, we provide the model with a batch of 100 examples, each example e_i consisting of 80 points, i.e., $e_i = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{80}, \mathbf{y}_{80})$. The nn-score is the attention paid to $\mathbf{y}_j$ for predicting $\mathbf{y}_k$ $(0 < j < k)$ where, $j = \arg\max_{i=0}^{k-1} \left(\frac{\mathbf{x}_i^T \mathbf{x}_k}{\|\mathbf{x}_i\| \|\mathbf{x}_k\|} \right)$ is the index of the nearest neighbour of $\mathbf{x}_k$, averaged for the last 40 points (i.e., $k > 40$) over all 100 examples. If the nn-score for a particular head is greater than 0.5, we label that head as a 'nearest neighbour head'. In our experiments with frozen GPT on the Nearest Neighbours task, we were able to consistently find 8-10 nearest neighbour heads across multiple runs.

4.5.2 Direct Evaluation Experiments

Details of the Setup. The main prompt (see Figure 4.11 for an example) comprises of an instruction in natural language followed by a series of k example points $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i \in \{0,1\}^d$ is provided as a sequence of tokens $x_1, \ldots, x_d$, $x_j \in \{0,1\}$ and $\mathbf{y}_i \in \{0,1\}$ is also provided as a single token. The model's goal is to predict the correct

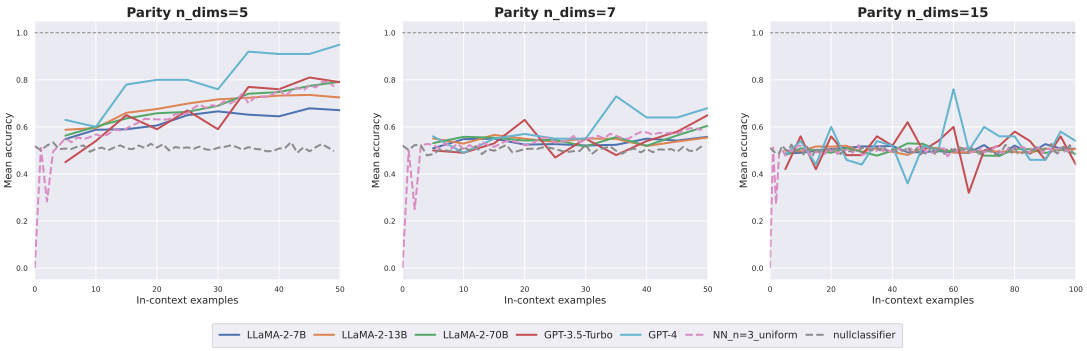


Figure 4.12: Results with the direct evaluation of LLMs at inference time across varying dimensions for the PARITY-n task.

label $\mathbf{y}_{k+1} \in \{0, 1\}$ for the query point $\mathbf{x}_{k+1}$. For a particular function, we sample a sequence of n points and prompt the model with $k < n$ points as in-context exemplars of the function. We call the model multiple times, increasing k by a value of 5 every time. The prompt in each call consists of the first k points ($5 < k < n$) from the n-point sequence, and asks to predict the $(k+1)^{\text{th}}$ point. We repeat this experiment with 100 different functions[2] from the same function class.

We experiment with three classes of functions: Conjunction, Majority, and Parity. We consider dimension $d \in \{5, 7, 15\}$ and set the number of points $n = 50$ when $d < 10$ and $n = 100$ when $d \geq 10$.

Results with Direct Evaluation. The performances of all LLMs for the Conjunction and Majority tasks with a varying number of dimensions are provided in Figure 4.13. It is clear that all models are able to perform as well as or better than the nearest neighbour baseline when the number of dimensions is up to 7. Note that in this setup, the LLM is essentially unaware of the task (such as Conjunction) when it is provided directly with the prompt at inference time. Even with $n = 7$, there are 2^{128} Boolean functions, and so the in-context learning problem remains challenging, and the observed performance of these models is impressive. It is perhaps surprising to see that the open-source LLaMA-2 model performs quite similar to GPT-3.5-Turbo. It is also particularly interesting to see the strong performance of GPT-4; apart from outperforming all other models, it also slightly outperforms the nearest neighbour baseline even in the 15-dimensional case.

The results of evaluating LLMs directly on the Parity task are provided in Figure 4.12. We also experimented with LLaMA-2 models of smaller scale as well as GPT-2 as

[2]It is prohibitively expensive to experiment with a very high number of functions for the OpenAI models, so we limit the number of functions to 100. We average over 2000 different functions for the baselines and 1000 different functions for the LLaMA-2 models for more robust results.

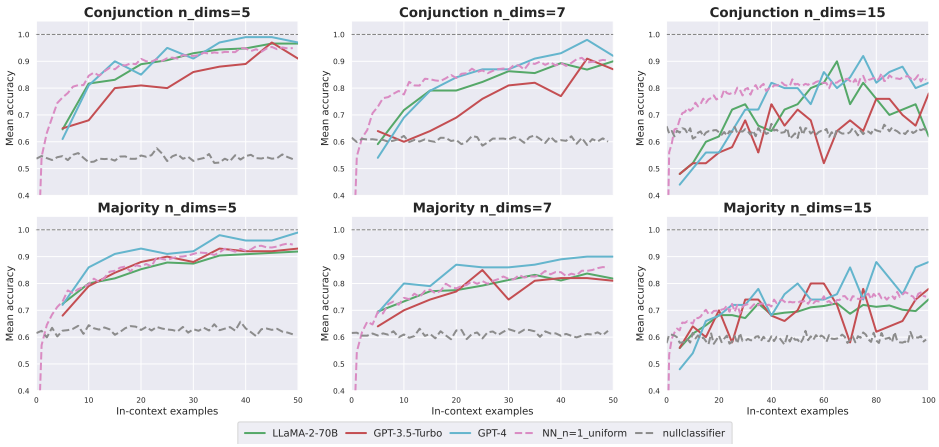


Figure 4.13: Results with the direct evaluation of LLMs at inference time. The *top* row shows the performance across varying dimensions for the Conjunction task while the *bottom* row shows the performance for the Majority task.

shown in Figure 4.14. Our results seem to indicate that model scale seems to play some role in the ability of LLMs to implement learning algorithms. This can be seen from the performance of the pre-trained GPT-2 model, which fails on both tasks even for 5 dimensions, and the (gradual) increase in performance as we increase the size of the LLaMA models. Also, it is quite surprising to see that even smaller LLaMA models are also able to achieve nontrivial levels of performance for both tasks.

Performance of Nearest Neighbours. In the case of sparse parities, nearest neighbour approaches can do quite well, particularly for relatively small values of k and n. For instance, suppose there are only two relevant bits, say $k = 2$, then for any fixed point $\mathbf{x}$, conditioned on the relevant bits matching, the expected distance to a random point is $(n-2)/2$. On the other hand, if the relevant bits don't exactly match, then the expected distance to a random point is at least $(n-2)/2+1$. Since the probability that the two relevant bits match is $1/4$, we expect with a relatively high probability that the nearest neighbour would match the relevant bits. As a result, the nearest neighbour classifier will have a reasonably high accuracy. We observe that the performance of LLMs on this task is better than random and comparable to nearest neighbour.

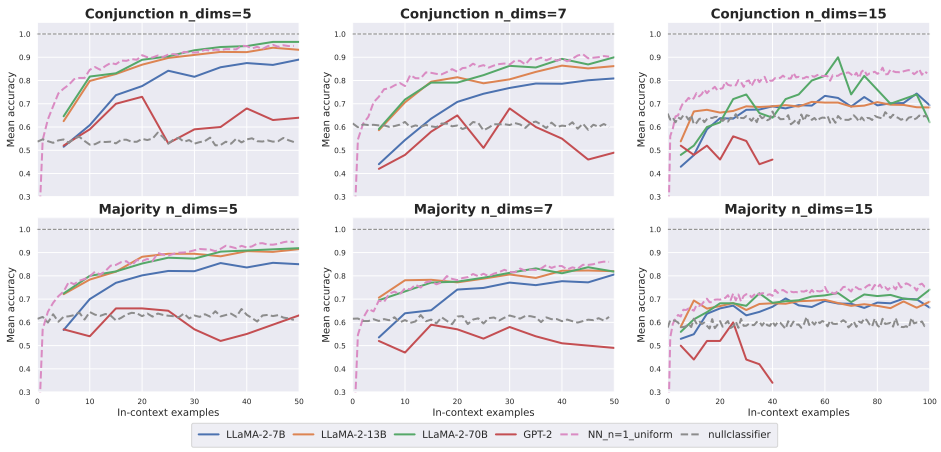


Figure 4.14: Results with direct evaluation for LLaMA models of different scales and GPT-2. The *top* row shows the performance across varying dimensions for the Conjunction task while the *bottom* row shows the performance for the Majority task.

4.6 Additional Experiments on Boolean Functions

4.6.1 Robustness of Models

In this section, we explore the robustness of Transformers to out-of-distribution (OOD) inputs and functions as well as their robustness to noisy examples during training.

OOD functions. Transformers seemingly perform in a near-optimal manner for tasks such as Conjunctions and Disjunctions. First, we evaluate the performance of Transformers trained on the Conjunctions task on functions from a different distribution than the ones seen during training. As described in Section 4.3.2, while training on the Conjunctions task, the functions are sampled in such a way that for any $i \in [n]$, the probability of the literal z_i or $\bar{z}_i$ being in the sampled Conjunctions is 30%. Hence, the expected number of literals in the Conjunctions seen during training is $n/3$. We evaluated Transformers trained on such Conjunctions on examples where the functions are sampled in such a way that the expected number of literals is $2n/3$ and the probability of any literal being in the sampled functions is twice as compared to the ones seen during training. We find that Transformers perform effectively on such examples as well (see Figure 4.15 Right). However, we find that the performance degrades for extreme changes in distribution where the expected number of literals in the functions is above 80%.

Trained on Threshold and Tested on Conjunctions. We also evaluated the performance of Transformers trained on 0-1 Threshold functions on the Conjunctions task. As describe in Section 4.3.1, 0-1 Threshold Functions are functions of the form

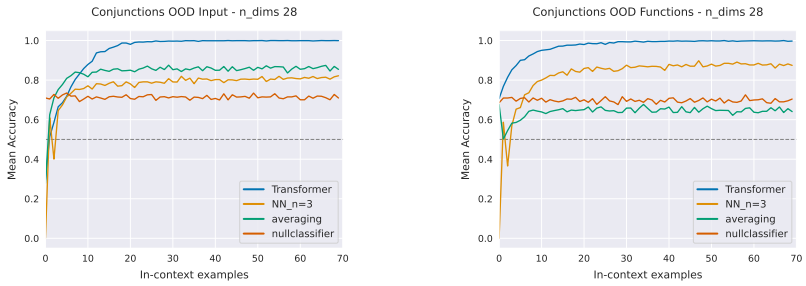


Figure 4.15: Performance of Transformers on Conjunctions with out-of-distribution inputs (left) and functions (right). Refer to Section 4.6.1 for more details.

Figure 4.16: Comparison between the performance of a Transformer trained to learn the 0-1 Threshold function on the Conjunction task (left) and the original 0-1 Threshold task. Refer to Section 4.6.1 for more details.

$\text{sign}(\sum_{i=1}^{n} w_i x_i - b)$ where $w_1, \ldots, w_n \in \{0, 1\}$ and $b \in \{-k, \ldots, k-1, k\}$. In some sense, Conjunctions are simpler threshold functions where the function is of the form $\text{sign}(\sum_{i=1}^{n} w_i x_i - (\|w\|_1 - 1))$. At the same time, note that the 0-1 Threshold functions task in our setting will not have functions which are exactly Conjunctions since $k = 3$ in our experiments and $\mathbb{E}[\|w\|_1] = n/3$ where n is set to 28 for experiments with Threshold functions.

We conduct an experiment where we train a Transformer on the 0-1 Threshold function task (with $n = 28$) and evaluate the model on the Conjunctions task. Perhaps surprisingly, we find that the model performs better on the Conjunctions task (achieving over 90% at the end) in comparison to the in-distribution Threshold function task (See Figure 4.16). It is interesting to see that the algorithm learned during training generalises to higher values of b in $\text{sign}(\sum_{i=1}^{n} w_i x_i - b)$. These observations could perhaps be attributed to the fact that Conjunctions are one of the simplest Threshold functions and could be easier since the value of b is dependent on w.

Robustness to noise. On tasks such as Conjunctions and Disjunctions where the models learn to perform in-context learning effectively, we explore whether they are robust to noisy labels during the training process. During training, for every example

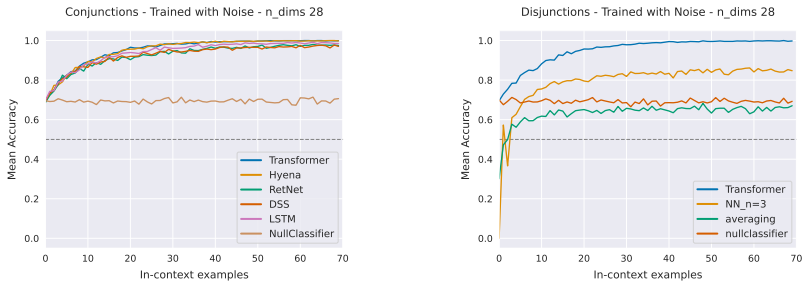


Figure 4.17: Left: Performance of various architectures trained with noisy examples on the Conjunctions task. Right: Performance of Transformers trained with noisy examples on the Disjunctions task and performance of baselines. Refer to Section 4.6.1 for more details.

$(\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_m, \mathbf{y}_m)$, we flip 10% of the labels $\mathbf{y}_i$s. Hence, the examples seen during training are not perfectly labeled by a Conjunction (or Disjunction). During the evaluation, we provide clean inputs (without noise) to test the performance of the models. We find that on these tasks, all architectures are robust to noise in the training process (see Figure 4.17).

Length Generalization. We explore the effectiveness of Transformers to generalize to a larger number of examples (in some sense length) in comparison to those seen during training. We consider Conjunctions and DNFs where they perform reasonably well in the in-distribution setting. For Conjunctions, we train the model on prompts with 60 examples and test their performance on prompts with 100 examples. Similarly, for DNFs, we train the model on prompts with 80 examples and test their performance on prompts with 160 (twice) examples. We compare two types of positional encodings: (a) absolute encodings and (b) no positional encodings with just causal masking. Figure 4.18 depicts the performance of Transformers on these tasks. Transformers with no positional encodings and just masking seem to generalize well whereas those with absolute positional encoding seem to struggle on higher lengths.

4.6.2 What determines the difficulty of in-context learning?

A natural question that arises from our results is: what properties of a task determine the difficulty of in-context learning for models such as Transformers and other architectures? First, it is interesting to observe that capacity measures such as VC dimension or the size of a function class do not seem to correlate well with the performance of the models (See Table 4.2). While Transformers are able to perform well on Conjunctions which have VC dimension $O(n)$ and sparse Disjunctions which have VC dimension $O(k \log n)$, they fail on Parities and sparse Parities with similar VC dimensions. VC

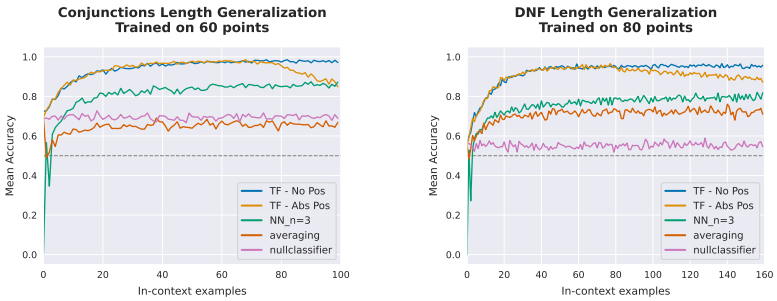


Figure 4.18: Performance of Transformers on prompts with a larger number of examples than seen during training. Left: Trained on Conjunctions with 60 examples and tested on 100. Right: Trained on DNFs with 80 examples and tested on 160. Transformers with no positional encodings and just masking generalize well whereas those with absolute encodings struggle on higher lengths. Refer to Section 4.6.1 for more details.

dimension is one way to measure the complexity of a class of functions which relates to the worst-case sample complexity required to learn functions from the class and it is related to several other measures which compute the capacity of the class in some manner.

Note that, measures such as VC dimension are not dependent on either the distribution of inputs or the functions. We explore some other measures in order to understand the properties that determine the difficulty of in-context learning. One measure which we found to correlate well with the performance of Transformers is the pairwise correlation between the functions in a class. For Boolean functions of the form $f : \{0,1\}^n \to \{\pm 1\}$, the pairwise correlation of a class of functions $\mathrm{P}(\mathcal{F})$ is defined as,

$$\mathrm{P}(\mathcal{F}) = \left| \mathbb{E}_{f_1, f_2}[\mathbb{E}_{x \sim U}[f_1(x) f_2(x)]] \right|. \tag{4.2}$$

The measure $\mathrm{P}(\mathcal{F})$ computes the magnitude of the expected correlation between pairs of functions sampled over $D_{\mathcal{F}}$. To estimate $\mathrm{P}(\mathcal{F})$ for the tasks considered in the chapter, we sample 1k functions according to the distribution used for training and sample 10k inputs from the uniform distribution over $\{0,1\}^n$. The estimated pairwise correlation of all function classes is provided in Table 4.2. See that the pairwise correlation between any two Parity functions is 0. We observe that the value $\mathrm{P}(\mathcal{F})$ is high for classes in which models perform well and decreases across categories with Parities having the minimum pairwise correlation.

While the pairwise correlation among functions has a reasonably strong correlation with the degradation of the performance of the models, it still has some weaknesses. As compared to properties of function classes such as VC dimension, it takes the

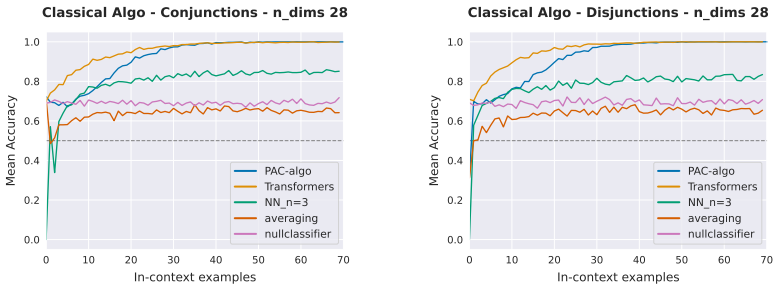


Figure 4.19: Comparison between the performance of classical PAC-learning algorithms for Conjunctions and Disjunctions and the performance of Transformers. See Section 4.3.5 for the description of the algorithm and other details.

distribution over functions into account. However, the input distribution is uniform over $\{0,1\}^n$ while computing $\mathrm{P}(\mathcal{F})$ whereas it is not uniform for some of the function classes during training. Having a clearer understanding of the factors that determine the difficulty of in-context learning is an interesting open problem.

Table 4.2: Pairwise correlations of different functions classes computed according to Eq. 4.2. See Section 4.6.2 for more details.

Tasks	Conjunctions	Disjunctions	Sparse Disjunctions	3-CNF	3-DNF	Majority	Integer-Threshold	Integer Halfspace	Parity-(10, 2)	Parity-20
VC Dimension	$O(n)$	$O(n)$	$O(k \log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(k \log n)$	$O(n)$
Pairwise Correlation	95.2 ± 0.07	95.3 ± 0.08	77.1 ± 0.01	51.86 ± 0.29	50.7 ± 0.26	20.7 ± 0.1	0.6 ± 0.21	0.1 ± 0.14	0.08 ± 0.03	0.01 ± 0.03

4.6.3 Exploring Sample Complexity

In this section, we explore the performance of Transformers when the number of examples during training is fixed. Recall that each training (or test) example $X = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_m, \mathbf{y}_m)$ is created by sampling m input points from the distribution D_X over inputs and sampling one target function from the distribution $D_{\mathcal{F}}$ over functions. For the experiments in Section 4.3, a fresh set of input points and functions are drawn during each training iteration.

For example, while training Transformers for 20k steps for the task Conjunctions with batch size 64, the model essentially observes about 1.3M functions and input sequences. However, note that the likelihood of observing the same function during evaluation is extremely low since for 28 dimensions, the total number of Conjunctions is over 22 trillion.

An interesting question is to understand how many examples Transformers need to generalize well in the in-context learning setting. Moreover, what is the effect of

Figure 4.20: Performance of Transformers on learning to learn Conjunctions with a finite number of examples. The figures show the change in performance across various axes such as depth, number of examples, and number of dimensions in the input. Refer to Section 4.6.3 for more details.

increasing the capacity of the model when the number of examples remains fixed? We explore these questions in this section.

We consider Conjunctions where Transformers are effective in the vanilla setting. During training, we first sample a fixed number of examples N of the form $(\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_m, \mathbf{y}_m)$ by sampling N functions from the distribution $D_{\mathcal{F}}$ and sampling N sequence of m input points from the distribution D_X. All training iterations over the N examples and during evaluation we sample fresh examples to estimate the accuracy of the model. We evaluate the model with $N \in \{128, 512, 1024, 2048, 4096, 8192, 16384\}$ examples. We test the models across tasks with dimensions $d \in \{14, 28, 42\}$. We find that neural networks such as Transformers and LSTMs are quite sample-efficient in terms of solving Conjunctions task where they achieve close to perfect accuracy even with 4096 examples. Figure 4.20 (left) and Figure 4.21 (left) depict the mean accuracy of Transformers and LSTMs across various number of examples N and task dimensions d. Note the mean accuracy during evaluation here is the mean of the accuracy of all m points in an example and not the accuracy on the last point. For instance, if we test on $K = 1280$ examples to estimate the accuracy and each example has $m = 70$ points then the mean accuracy is computed as $\frac{1}{K}\sum_{i=1}^{K}\left(\frac{1}{m}\sum_{k=1}^{m}\mathbb{I}[M(P_k^{(i)}) = f_i(\mathbf{x}_k)]\right)$. The mean accuracy across all in-context examples will always be lower than 100% since the model will make mistakes in the first half of the examples before converging to a near-perfect accuracy for tasks like Conjunctions.

It is also interesting to see that increasing the capacity of the model has a negligible effect on the performance of the model. Figure 4.20 (right) shows the mean accuracy of Transformers across different depths (number of layers) when trained on 4096 examples.

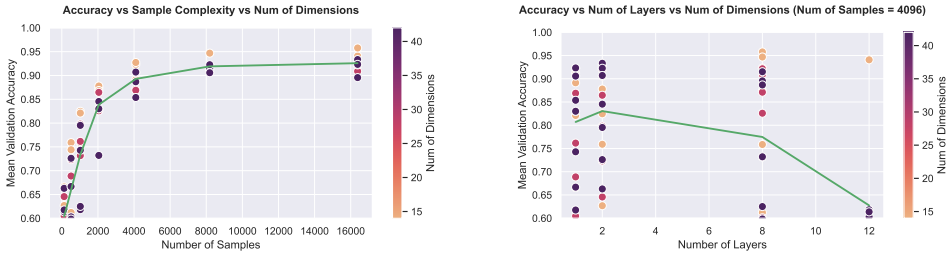


Figure 4.21: Performance of LSTMs on learning to learn Conjunctions with a finite number of examples. The figures show the change in performance across various axes such as depth, number of examples, and number of dimensions in the input. Refer to Section 4.6.3 for more details.

4.6.4 Implementation Details

All of our implementations for experiments are based on PyTorch [121]. For our experiments with Transformers trained from scratch, we use the Huggingface Transformers library [171] and our own custom implementation. For recurrent models such as LSTMs, we use PyTorch's inbuilt implementation. For Hyena [127] and DSS [67], we adapt the official implementation by the authors for our experimental setting. For RetNet [149], we use our own implementation.

All our experiments were conducted using 16 NVIDIA Tesla V100 GPUs each with 16GB memory. For each dataset, we extensively tune across several hyperparameters and report the results based on the best-performing models. Table 4.3 lists the hyperparameters used for tuning the models for Boolean function experiments in Section 4.3. We use a grid search procedure to tune the hyperparameters. For all our results, we used Adam Optimizer and tuned the learning rates. For all architectures apart from LSTMs, we tuned across depths $\in \{1, 2, 6, 12, 16\}$ and for LSTMs we used depths $\{1, 2, 3, 8\}$ primarily because LSTMs with large depths are harder to train. For LSTMs we tuned the width or hidden_size across $\{64, 256, 378, 512\}$ and for other architectures we use $\{256, 512\}$. For Hyena, we tune the order hyperparameter across $\{2, 3\}$ and for Transformers and Retnets we tune the heads across $\{4, 8\}$. For all non-recurrent models, we try both learnable and absolute positional embedding schemes. For all models, we tune the learning rate across $\{0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.000001\}$. We train the models for different numbers of steps/iterations $\in$ [30k, 500k] depending on the task where each iteration is with a fresh batch of in-context sequences. Note that in all these tasks apart from the experiments with finite samples in Section 4.6.3, the training loss never reduces to 0 since each batch of examples is labeled by a different function (with high probability). We train for a large number of steps (200k)

Hyperparameter	Transformer and others	LSTM
D_model/Hidden Size	[256, 512]	[64, 512]
Heads	[4, 8]	-
Order	[2, 3]	-
Number of Layers	[1, 16]	[1, 8]
Learning Rate	[1e-2, 1e-5]	[1e-2, 1e-5]
Position Encoding Scheme	[Learnable, Absolute]	-

Table 4.3: Different hyperparameters and the range of values considered for each of them. The hyperparameter *Heads* is only relevant for Transformers and RetNets. The hyperparameter *Order* is only relevant for Hyena. See Section 4.6.4 for more details.

if the loss doesn't reduce or train till the loss has reduced and converged for at least 20k steps. For instance, for tasks like Conjunctions models converge to low loss in 20k steps so we train the models to up to 40k steps. For other tasks such as Parities, we train for 500k steps. For the additional linear regression task (Figure 4.4), we follow the same setup and hyperparameters as described in [57].

Based on the experiments, we find that Transformers and Hyena are relatively much more robust across various hyperparameters than other architectures. Among most tasks and architecture, we find that the learning rate has the most influence on the performance and the depth has a nontrivial influence. For other hyperparameters such as heads and widths, we do not see any nontrivial difference in performance.

Direct Evaluation Experiments. For experiments with open-source models such as LLaMA-2, we use Huggingface Text-Generation-Inference[3]. Experiments using GPT-3.5-Turbo and GPT-4 were performed using the OpenAI API[4]. Experiments with locally deployed large models like LLaMA-2 were conducted using 2 A100 GPUs each with 80GB memory. For all experiments, we decode greedily and generate a maximum of two tokens (whitespace and the label[5]).

[3] https://github.com/huggingface/text-generation-inference

[4] https://platform.openai.com/

[5] If the model does not predict a label in $\{0, 1\}$, we consider the prediction incorrect.

4.7 Conclusion

In this chapter, we highlighted certain limitations of Transformers in in-context learning and their ability to leverage more informative examples, such as teaching sequences for learning in a more sample-efficient manner. Additionally, we showed that pretrained models also learn mechanisms to solve tasks in the stylised ICL setting. Further, we show that LLMs are also capable of performing as well as certain baseline learning algorithms in a more practical setting. Our results raise several interesting questions for future work: (a) Can LLMs leverage more informative examples to sample-efficiently learn practical tasks in-context? (b) What aspects of the pretraining process of LLMs lead to their ability to implement learning algorithms since they are not primarily trained in the meta-learning-like framework? (c) What are the theoretical limitations behind the difficulty of learning Parities in the in-context learning setting? (d) What are the mechanisms that various architectures (attention, recurrence, and convolution) use to learn tasks using different operations?

Chapter 5

Learning Attention with Queries

5.1 Introduction

We now turn to the theoretical learnability of our model classes. Rather than learning from random examples as in previous chapters, we consider richer supervision settings, such as access to value and membership queries. In this chapter, we study the learnability of attention-based models with queries, focusing on proper learning, where the target and hypothesis lie in the same class. In the subsequent chapter, we study the learnability of automata from various query models and apply them to more realistic Transformer-based language models.

Transformer-based models [161] are being widely deployed in the industry. Given their ubiquity, a natural question is the following: given blackbox access to the outputs of a target model (e.g., via an API), can an adversary recover the weights of the target model? Such questions regarding model stealing have been widely studied empirically and theoretically for feedforward and related networks [158, 140, 77, 115]. A natural formalisation of this problem is the problem of *learning with queries* [29, 8] where a learner adaptively chooses inputs and observes labels (or real-valued outputs) with the goal of reconstructing the target function or, more stringently, the target parameters. Beyond security-driven model extraction, parameter recovery is also a natural lens for understanding what aspects of a model are identifiable from behaviour alone and which architectural components admit efficient reconstruction.

A strand of work has focused on providing provable guarantees for recovering ReLU networks under value or membership queries, often under additional structural assumptions [35, 105, 45]. In parallel, there is a body of work that studies simplified attention models and one-layer Transformers, establishing learnability results or characterising training dynamics under random examples [154, 97, 164, 99, 9]. However,

parameter recovery guarantees for softmax-attention from value queries have been largely underexplored.

Problem. We study parameter recovery for attention-based sequence models with *value-query* access. Concretely, we begin with a single-head attention-based regressor which can be formulated in the following way (See Sec. 5.2 for details). For an input sequence $X \in \mathbb{R}^{N\times d}$ (with variable length $N \geq 1$) and parameters $\boldsymbol{W} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{v} \in \mathbb{R}^d$, the model computes

$$f_{\boldsymbol{W},\boldsymbol{v}}(X) = \mathsf{softmax}\big(\boldsymbol{x}_1^\top \boldsymbol{W}\boldsymbol{x}_N, \ldots, \boldsymbol{x}_N^\top \boldsymbol{W}\boldsymbol{x}_N\big)^\top (X\boldsymbol{v}) \in \mathbb{R},$$

where the last token plays the role of the query and the softmax produces attention weights over positions. A learner receives black-box access to $\mathrm{VQ}(X) = f_{\boldsymbol{W}^\star,\boldsymbol{v}^\star}(X)$ and can query *any* sequence X; the goal is to recover $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$ exactly or approximately.

Contributions. We initiate a systematic study of learning softmax attention with value queries and provide the first, to our knowledge, provable guarantees for parameter recovery. We analyse learnability in different query models and regimes, and establish the following results in this chapter.

(i) Learning single-layer attention and Transformers. We give an elementary, polynomial-time algorithm that exactly recovers $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$ for the single-head attention regressor using $O(d^2)$ value queries (Theorem 9). The key observation is that we can use short sequence vectors to isolate the softmax nonlinearity. Using queries of length two reduces softmax to a sigmoid, which can be inverted, turning each oracle response into linear equations that can be solved to obtain the target parameters. Further, we show that the single-head algorithm can be lifted to learn a one-layer, single-head Transformer that composes attention with a two-layer ReLU MLP: assuming an algorithm exists for learning the corresponding ReLU FFN from value queries, we combine it with our attention-recovery method to obtain a value-query learner for the Transformer (Section 5.3.1).

(ii) Faster recovery in low-rank regime. Motivated by the common setting where the head dimension $r \ll d$, we analyse the case $\mathrm{rank}(\boldsymbol{W}^\star) \leq r$. We give a randomized algorithm that recovers $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$ using $O(rd)$ queries (Theorem 11). The main idea is to design probes that implement rank-one linear measurements of $\boldsymbol{W}^\star$, reducing recovery to a standard low-rank matrix sensing problem and enabling reconstruction via compressed sensing tools.

(iii) Robustness and Stability. Our exact recovery algorithm relies on inverting a sigmoid, which is not globally Lipschitz. We therefore study a more realistic oracle that returns values within additive tolerance τ. Under mild norm bounds and a simple

margin condition, we show that one can recover $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$ to ϵ accuracy in polynomial time with $O(d^2)$ queries and with tolerances scaling polynomially in d and the target accuracy (Section 5.5). The proof constructs probes that keep the recovered attention weights in a stable range where the inverse-sigmoid behaves well.

Lastly, we show that multi-head attention parameters are *not identifiable* from value queries in general: distinct parameter settings can induce exactly the same input–output map (Proposition 13). Consequently, guarantees analogous to the single-head case cannot hold without additional assumptions. We discuss some structural conditions that allow identifiability and outline possible approaches in this regime (Section 5.6).

At a high level, our first main result shows that single-head softmax attention admits surprisingly straightforward parameter recovery. This stands in contrast to learning with random examples (without queries), where the problem is non-convex for standard loss functions and is quite challenging. Further, the analogous problem of learning a one-hidden-layer ReLU MLP with queries, which is similar in surface form, has also been substantially more challenging [35]. Taken together, our results clarify what value queries can and cannot reveal about attention-based models. In the idealized query setting, single-head softmax attention is fully and efficiently reconstructible, and low-rank structure further reduces query complexity. At the same time, the multi-head setting exhibits non-identifiability barriers, suggesting that provable extraction for realistic attention blocks will require additional structure or weaker learning goals.

5.2 Definitions

Each example consists of a matrix–label pair (X, y) where $X \in \mathbb{R}^{N\times d}$ contains N row-vectors $X = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{pmatrix}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, and the target label is a scalar $y \in \mathbb{R}$.

Model class Att. Fix an integer $d \geq 1$ and the model can take inputs of any length $N \geq 1$. We focus on a single attention-based regressor that takes a sequence of vectors as input and produces a scalar as output. Let $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{r\times d}$ be the query, key, and value projection matrices. Let $\boldsymbol{w}_o \in \mathbb{R}^r$ be the output projection vector. For any N, the output of a single attention-based model $\mathrm{SHA} : \mathbb{R}^{N\times d} \to \mathbb{R}$ is computed by,

$$\mathrm{SHA}(X) = \mathsf{softmax}(\boldsymbol{x}_1^\top \boldsymbol{K}^\top \boldsymbol{Q} \boldsymbol{x}_N, \ldots, \boldsymbol{x}_N^\top \boldsymbol{K}^\top \boldsymbol{Q} \boldsymbol{x}_N)^\top (X \boldsymbol{V}^\top \boldsymbol{w}_o)$$

The model can be viewed in a simpler way in which the query and key projections are merged into one matrix, i.e., $\boldsymbol{x}_i \boldsymbol{K}^\top \boldsymbol{Q} \boldsymbol{x}_N = \boldsymbol{x}_i \boldsymbol{W} \boldsymbol{x}_N$ for some matrix $\boldsymbol{W} \in \mathbb{R}^{d \times d}$. Additionally, the value projection and output vector can be merged: $X \boldsymbol{V}^\top \boldsymbol{w}_o = X \boldsymbol{v}$ for some vector $\boldsymbol{v} \in \mathbb{R}^d$. More precisely, an equivalent definition is as follows. For parameter vectors $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{v} \in \mathbb{R}^d$ define the score vector

$$s_i(X, \boldsymbol{W}) = \boldsymbol{x}_i^\top \boldsymbol{W} \boldsymbol{x}_N, \qquad i = 1, \ldots, N,$$

and the (row) attention weights

$$\boldsymbol{\alpha}(X, \boldsymbol{W}) = \mathsf{softmax}\big(s_1, \ldots, s_N\big) \in \Delta^{N-1},$$

where $\mathsf{softmax}(z)_i = \exp(z_i) / \sum_j \exp(z_j)$ and Δ^{N-1} is the probability simplex. The attention model associated with $(\boldsymbol{W}, \boldsymbol{v})$ is the map

$$f_{\boldsymbol{W}, \boldsymbol{v}}(X) = \boldsymbol{\alpha}(X, \boldsymbol{W})^\top (X \boldsymbol{v}) \in \mathbb{R}.$$

With all such functions, we have the hypothesis class

$$\textsc{Att}_d = \big\{ f_{\boldsymbol{W}, \boldsymbol{v}} \big| \boldsymbol{W} \in \mathbb{R}^{d \times d} \text{ and } \boldsymbol{v} \in \mathbb{R}^d \big\}.$$

The class $\textsc{Att} = \cup_{d \geq 1} \textsc{Att}_d$. The single-head attention-based model considered here is essentially a one-layer Transformer with a linear network instead of a ReLU MLP.

Multi-head Attention. For positive integers H and d_h, let $d = H d_h$. We will have H parameter matrices $\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(H)} \in \mathbb{R}^{d \times d}$ with rank $\leq d_h$ where $\boldsymbol{W}^{(h)} = (\boldsymbol{K}^{(h)})^\top \boldsymbol{Q}^{(h)}$ similar to the single head definition. We will have H value projection matrices $\boldsymbol{V}^{(1)}, \ldots, \boldsymbol{V}^{(H)} \in \mathbb{R}^{d_h \times d}$ and one output projection vector $\boldsymbol{w}_o^{d \times 1}$. The hth attention weight will then be computed by

$$\boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)}) = \mathsf{softmax}(\boldsymbol{x}_1^\top \boldsymbol{W}^{(h)} \boldsymbol{x}_N, \ldots, \boldsymbol{x}_N^\top \boldsymbol{W}^{(h)} \boldsymbol{x}_N) \in \Delta^{N-1}$$

and the output of the hth head will be,

$$\boldsymbol{a}^{(h)} = \boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)})^\top (X (\boldsymbol{V}^{(h)})^\top) \in \mathbb{R}^{d_h}.$$

Following the standard definition, the outputs are then concatenated to compute the final output,

$$\boldsymbol{a} = [\boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(H)}] \in \mathbb{R}^{d \times 1} \quad f_\theta^H(X) = \boldsymbol{a}^\top \boldsymbol{w}_o \in \mathbb{R},$$

where θ has all the parameter matrices $\boldsymbol{W}^{(h)}, \boldsymbol{V}^{(h)}$ for all h and the vector $\boldsymbol{w}_o$. Note that we can merge the value matrices and output projection matrices in the following

way. Partition $\boldsymbol{w}_o$ in H equal and contiguous parts $\boldsymbol{w}_o = [\boldsymbol{w}_o^{(1)}, \ldots, \boldsymbol{w}_o^{(H)}]$ where $\boldsymbol{w}_o^{(h)} \in \mathbb{R}^{d_h}$. Then for vectors $\boldsymbol{v}^{(h)} = (\boldsymbol{V}^{(h)})^\top \boldsymbol{w}_o^{(h)} \in \mathbb{R}^{d\times 1}$, we can rewrite,

$$f_\theta^H(X) = \sum_{h=1}^{H} \boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)})^\top (X(\boldsymbol{V}^{(h)})^\top)\boldsymbol{w}_o^{(h)} = \sum_{h=1}^{H} \boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)})^\top (X\boldsymbol{v}^{(h)}).$$

Thus, the multi-head attention model can be seen as a sum of H single-head attention models with their own parameters $(\boldsymbol{W}^{(h)}, \boldsymbol{v}^{(h)})$.

Learning objective. Throughout, we primarily focus on parameter estimation from value or membership queries. By *learning*, we will mean exact or approximate identification of the target parameters. Given black-box access to the target function, the goal of the learner is to identify the target either exactly or approximately by querying the target function. For instance, for a target function $f^\star$ parameterised by $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$, the learner is given access to a value query oracle $\mathrm{VQ}(X) = f^\star(X)$ which returns the true output exactly (or in some cases approximately as in Sec. 5.5). The learner can query any input $X \in \mathcal{X}$ to the value query oracle and the learner aims to recover $(\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{v}})$ such that $(\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{v}}) = (\boldsymbol{W}^\star, \boldsymbol{v}^\star)$, or,

$$\|\boldsymbol{W}^\star - \widehat{\boldsymbol{W}}\|_F \leq \epsilon \quad \text{and} \quad \|\boldsymbol{v}^\star - \widehat{\boldsymbol{v}}\|_2 \leq \epsilon.$$

A class of functions is efficiently learnable if the number of queries used and the runtime of the learning algorithm are polynomial in the parameters of the concept class (such as d for the class Att).

5.3 Learning Single-Head Attention with Queries

We will analyse whether an unknown single-head attention-based regressor can be learned exactly using value queries only. The learner has access to the value query oracle $f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star} : \mathbb{R}^{N\times d} \to \mathbb{R}$ such that for any $X \in \mathbb{R}^{N\times d}$, it can query the label $f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star}(X) \in \mathbb{R}$. We describe a value query (VQ) method that exactly identifies the target parameters $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$ whenever $\boldsymbol{v}^\star \neq \mathbf{0}$. Throughout, let $\mathbf{e}_i \in \mathbb{R}^d$ denote the ith standard basis vector, and let $\sigma(t) = 1/(1 + e^{-t})$ denote the sigmoid function.

Theorem 9. *Assuming $\boldsymbol{v}^\star \neq \mathbf{0}$, given access to a value query oracle $\mathrm{VQ}(X) = f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star}(X)$, the parameters $\boldsymbol{W}^\star$ and $\boldsymbol{v}^\star$ are exactly learnable in polynomial-time with $O(d^2)$ value queries.*

Proof. The algorithm has two phases. The first phase is trivial, and recovers $\boldsymbol{v}^\star$ using one-row or length-one queries. The second phase recovers $\boldsymbol{W}^\star$ column–by–column using two–row or length-two queries.

(i) Identifying $\boldsymbol{v}^\star$. With $N = 1$ the softmax weight is 1, so $f_{\boldsymbol{W},\boldsymbol{v}}([\boldsymbol{x}^\top]) = \boldsymbol{x}^\top \boldsymbol{v}$ and is independent of $\boldsymbol{W}$. Query the d one–row inputs $X = [\mathbf{e}_i^\top]$ for $i = 1, \ldots, d$ to obtain $y = \boldsymbol{v}_i^\star$. Hence, the vector $\boldsymbol{v}^\star$ is recovered with d VQs.

(ii) Identifying $\boldsymbol{W}^\star$ (column–wise recovery). Fix $j \in \{1, \ldots, d\}$ and let $\mathbf{w}_j := \boldsymbol{W}^\star \mathbf{e}_j \in \mathbb{R}^d$ and $(\mathbf{w}_j)_i = \boldsymbol{W}_{ij}^\star$. For any probe vector $\boldsymbol{u} \in \mathbb{R}^d$, consider the two–row input $X = \begin{bmatrix} (\boldsymbol{u} + \mathbf{e}_j)^\top \\ \mathbf{e}_j^\top \end{bmatrix}$. The attention scores are

$$s_1 = (\boldsymbol{u} + \mathbf{e}_j)^\top \boldsymbol{W}^\star \mathbf{e}_j = (\boldsymbol{u} + \mathbf{e}_j)^\top \mathbf{w}_j, \qquad s_2 = \mathbf{e}_j^\top \boldsymbol{W}^\star \mathbf{e}_j = (\mathbf{w}_j)_j.$$

So the attention weight on the first row or position is

$$\alpha(\boldsymbol{u}; j) = \frac{e^{s_1}}{e^{s_1} + e^{s_2}} = \sigma(s_1 - s_2) = \sigma\big(\boldsymbol{u}^\top \mathbf{w}_j\big), \tag{5.1}$$

where $\sigma(t) = 1/(1 + e^{-t})$. Since $(X\boldsymbol{v}^\star) = \left[\boldsymbol{u}^\top \boldsymbol{v}^\star + \boldsymbol{v}_j^\star, \boldsymbol{v}_j^\star\right]^\top$, the label returned by the oracle is the convex combination

$$y = \alpha(\boldsymbol{u}; j)\left((\boldsymbol{u} + \mathbf{e}_j)^\top \boldsymbol{v}^\star\right) + \big(1 - \alpha(\boldsymbol{u}; j)\big)\, \boldsymbol{v}_j^\star = \boldsymbol{v}_j^\star + \alpha(\boldsymbol{u}; j)\,(\boldsymbol{u}^\top \boldsymbol{v}^\star).$$

If $\boldsymbol{u}^\top \boldsymbol{v}^\star \neq \mathbf{0}$, we can recover the attention weight from the observed y via

$$\alpha(\boldsymbol{u}; j) = \frac{y - \boldsymbol{v}_j^\star}{\boldsymbol{u}^\top \boldsymbol{v}^\star} \in (0, 1).$$

Thus, combining with equation 5.1, we have,

$$\boldsymbol{u}^\top \mathbf{w}_j = \sigma^{-1}\left(\frac{y - \boldsymbol{v}_j^\star}{\boldsymbol{u}^\top \boldsymbol{v}^\star}\right)$$

Thus, each probe vector $\boldsymbol{u}$ yields a *linear* equation in the unknown column $\mathbf{w}_j$. This leads to one linear equation for d variables in $\mathbf{w}_j$. We can use d linearly independent vectors to identify $\mathbf{w}_j$ by solving a system of linear equations.

Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d$ be d linearly independent vectors such that $\boldsymbol{u}_\ell^\top, \boldsymbol{v}^\star \neq \mathbf{0}$ for all $\ell \in [d]$. Take invertible $\boldsymbol{Z} = [\boldsymbol{u}_1^\top; \ldots; \boldsymbol{u}_d^\top] \in \mathbb{R}^{d \times d}$ and collect d measurements $\boldsymbol{t} = [t_1, \ldots, t_d]$ with $\boldsymbol{Z}\mathbf{w}_j = \boldsymbol{t}$. One can recover the jth column of $\boldsymbol{W}^\star$ with $\mathbf{w}_j = \boldsymbol{Z}^{-1}\boldsymbol{t}$ and repeating this for $j = 1, \ldots, d$ recover $\boldsymbol{W}^\star$.

One could sample d vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d$ i.i.d. from $\mathcal{N}(0, I_d)$ and they will be linearly independent and the inner product $\boldsymbol{u}_\ell^\top \boldsymbol{v}^\star \neq 0$ for all ℓ almost surely. Another intuitive

approach to understand how the parameters are recovered is the following. Pick any index p with $\boldsymbol{v}_p^\star \neq 0$. For $\ell = 1, \ldots, d$, set the probe vector $\boldsymbol{u}_\ell$ in the following way,

$$\boldsymbol{u}_\ell = \begin{cases} \mathbf{e}_\ell, & \text{if } \boldsymbol{v}_\ell^\star \neq 0, \\ \mathbf{e}_\ell + \mathbf{e}_p, & \text{if } \boldsymbol{v}_\ell^\star = 0. \end{cases}$$

For any column vector $\mathbf{w}_j$, it is easy to see that the measurements $t_\ell = \boldsymbol{u}_\ell^\top \mathbf{w}_j = \mathbf{w}_{\ell j}$ when $\boldsymbol{v}_\ell^\star \neq 0$ (e.g. for $\ell = p$) and $t_\ell = \mathbf{w}_{\ell j} + \mathbf{w}_{pj}$ when $\boldsymbol{v}_\ell^\star = 0$.
The first phase uses d queries to recover $\boldsymbol{v}^\star$ and the second phase uses d VQs per column, totalling d^2 additional queries. Hence, the entire procedure uses $O(d^2)$ VQs and runs in polynomial time. If $\boldsymbol{v}^\star = \mathbf{0}$, then $f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star} \equiv 0$ for all X and $\boldsymbol{W}^\star$ is not identifiable from VQs (many $\boldsymbol{W}$ induce the same function). Conversely, when $\boldsymbol{v}^\star \neq \mathbf{0}$, the column–wise linear system above uniquely determines $(\boldsymbol{W}^\star, \boldsymbol{v}^\star)$.

□

Discussion. Our result indicates that the problem of learning a single-head attention model with value queries is relatively straightforward. It is worth understanding why this is somewhat surprising. First, in contrast, proving comparable guarantees from i.i.d. random examples (i.e., without queries) for attention-style models typically requires nontrivial assumptions on the data model, task, or training dynamics, and substantial effort has gone into understanding learnability and establishing such results even in restricted settings [164, 154, 9, 99, 97]. Similar to the problem of learning a single-head attention model, the problem of learning a one-hidden-layer ReLU MLP also has a single nonlinearity together with a parameter matrix and a vector. However, this analogous problem appears substantially more challenging: under minimal assumptions, existing efficient guarantees are generally distribution-dependent and focus on approximate function recovery (e.g., under Gaussian marginals), while exact recovery typically requires additional structural assumptions; see Chen et al. [35] for further discussion. Lastly, it is worth noting that many elementary concept classes, such as conjunctions or singletons, are not learnable with a polynomial number of queries alone.

5.3.1 Learning 1-Layer Transformers

In this section, we show how to learn a one-layer Transformer with single-head attention, assuming a value-query learner for two-layer ReLU MLPs exists. In particular, given an algorithm $\mathcal{A}_{\mathrm{FFN}}$ that learns two-layer ReLU MLPs from value queries, we give a

method that combines $\mathcal{A}_{\mathrm{FFN}}$ with our algorithm for single-head attention to learn a one-layer, single-head Transformer.

Model and Setup. A one-layer Transformer with single-head attention is the composition of a ReLU MLP and a single-head attention block. The attention block produces a single vector (here we attend using the last token as the query), and the FFN maps $\mathbb{R}^d \to \mathbb{R}$. Concretely, let $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{r\times d}$ and $\boldsymbol{V} \in \mathbb{R}^{d\times d}$ denote the query, key, and value projections, and let the FFN be a two-layer ReLU network $\mathrm{FFN}(z) = \boldsymbol{w}_2^\top \mathrm{ReLU}(\boldsymbol{W}_1 z)$ with $\boldsymbol{W}_1 \in \mathbb{R}^{m\times d}$ and $\boldsymbol{w}_2 \in \mathbb{R}^m$. We assume the FFN has no bias terms. Then for $X \in \mathbb{R}^{N\times d}$ the model can be written as

$$\mathrm{TF}(X) \;=\; \boldsymbol{w}_2^\top \mathrm{ReLU}\Big(\boldsymbol{W}_1 \cdot \big[\mathsf{softmax}(\boldsymbol{x}_i^\top \boldsymbol{K}^\top \boldsymbol{Q} \boldsymbol{x}_N)^\top (X\boldsymbol{V}^\top)\big]\Big).$$

As in Section 5.2, we merge query and key parameters by setting $\boldsymbol{W} := \boldsymbol{K}^\top \boldsymbol{Q} \in \mathbb{R}^{d\times d}$, so the scores are $s_i(X, \boldsymbol{W}) = \boldsymbol{x}_i^\top \boldsymbol{W} \boldsymbol{x}_N$ and $\boldsymbol{\alpha}(X, \boldsymbol{W}) = \mathsf{softmax}(s_1, \ldots, s_N)$. We also merge the value map and the first FFN layer by defining $\boldsymbol{A} := \boldsymbol{V}^\top \boldsymbol{W}_1^\top \in \mathbb{R}^{d\times m}$, and rename the second-layer vector $\boldsymbol{w}_o := \boldsymbol{w}_2$. With these reparameterisations,

$$\mathrm{TF}(X) = \boldsymbol{w}_o^\top \mathrm{ReLU}\big(\boldsymbol{\alpha}(X, \boldsymbol{W})^\top X \boldsymbol{A}\big),$$

which is the same architecture with the three parameters $(\boldsymbol{W}, \boldsymbol{A}, \boldsymbol{w}_o)$. We assume access to a value-query oracle $\mathrm{VQ}(X) = \mathrm{TF}_{\boldsymbol{W}^\star, \boldsymbol{A}^\star, \boldsymbol{w}_o^\star}(X)$.

Using an FFN learner. We will reduce learning TF to (i) learning a two-layer ReLU network from black-box queries and (ii) learning single-head attention as in Theorem 9. Formally, assume there exists an algorithm $\mathcal{A}_{\mathrm{FFN}}$ that, given oracle access to a function of the form $x \mapsto \boldsymbol{w}_o^{\star\top} \mathrm{ReLU}(x^\top \boldsymbol{A}^\star)$ (no biases), outputs parameters $(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{w}}_o)$ computing the *same* function on all $x \in \mathbb{R}^d$. To invoke $\mathcal{A}_{\mathrm{FFN}}$, we use one-row queries. When $N = 1$, the softmax weight is 1, and the attention output equals its single token, hence for $X = [x^\top]$ we have the scalar identity $\mathrm{VQ}([x^\top]) = \boldsymbol{w}_o^{\star\top} \mathrm{ReLU}(x^\top \boldsymbol{A}^\star)$. Therefore, restricting the Transformer oracle to length-1 inputs yields exactly the FFN target needed by $\mathcal{A}_{\mathrm{FFN}}$, and we can recover an equivalent pair $(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{w}}_o)$.

Recovering parameters of attention. We next show that $\boldsymbol{W}^\star$ can be recovered independently of $\mathcal{A}_{\mathrm{FFN}}$ by eliminating the ReLU using two queries. Define $\widetilde{\mathrm{VQ}}(X) := \mathrm{VQ}(X) - \mathrm{VQ}(-X)$. For softmax attention, negating all tokens does not change the scores because $s_i(-X, \boldsymbol{W}) = (-\boldsymbol{x}_i)^\top \boldsymbol{W}(-\boldsymbol{x}_N) = s_i(X, \boldsymbol{W})$, and thus $\boldsymbol{\alpha}(-X, \boldsymbol{W}) = \boldsymbol{\alpha}(X, \boldsymbol{W})$. Consequently,

$$\boldsymbol{\alpha}(-X, \boldsymbol{W})^\top (-X)\boldsymbol{A} \;=\; -\,\boldsymbol{\alpha}(X, \boldsymbol{W})^\top X \boldsymbol{A}.$$

Using $\mathrm{ReLU}(u) - \mathrm{ReLU}(-u) = u$ coordinate-wise, we obtain

$$\widetilde{\mathrm{VQ}}(X) \;=\; \boldsymbol{w}_o^{\star\top}\Big(\mathrm{ReLU}(z) - \mathrm{ReLU}(-z)\Big) \;=\; \boldsymbol{w}_o^{\star\top} z \;=\; \boldsymbol{\alpha}(X, \boldsymbol{W}^\star)^\top X\, \boldsymbol{v}^\star,$$

$$z = \boldsymbol{\alpha}(X, \boldsymbol{W}^\star)^\top X \boldsymbol{A}^\star,$$

where $\boldsymbol{v}^\star := \boldsymbol{A}^\star \boldsymbol{w}_o^\star \in \mathbb{R}^d$. Thus $\widetilde{\mathrm{VQ}}$ is *exactly* a value-query oracle for the single-head attention regressor $f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star}(X) = \boldsymbol{\alpha}(X, \boldsymbol{W}^\star)^\top (X\boldsymbol{v}^\star)$ considered in Section 5.3.1. Whenever $\boldsymbol{v}^\star \neq \boldsymbol{0}$, Theorem 9 applied to $\widetilde{\mathrm{VQ}}$ recovers $\boldsymbol{W}^\star$ (and $\boldsymbol{v}^\star$) exactly; each query to $\widetilde{\mathrm{VQ}}$ uses two regular value queries.

Together, the two steps yield a learning algorithm for one-layer single-head Transformers: use length-1 queries and $\mathcal{A}_{\mathrm{FFN}}$ to recover an equivalent FFN $(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{w}}_o)$, and use antisymmetric queries $\widetilde{\mathrm{VQ}}$ together with Theorem 9 to recover $\widehat{\boldsymbol{W}} = \boldsymbol{W}^\star$. If $\boldsymbol{v}^\star = \boldsymbol{A}^\star \boldsymbol{w}_o^\star = \boldsymbol{0}$, then $\widetilde{\mathrm{VQ}} \equiv 0$ and $\boldsymbol{W}^\star$ is not identifiable from such queries. The overall query complexity is $Q_{\mathrm{FFN}}(d, m) + O(d^2)$ up to constant factors, where $Q_{\mathrm{FFN}}(d, m)$ is the number of black-box queries used by $\mathcal{A}_{\mathrm{FFN}}$.

Discussion. Several prior works have studied the learnability of ReLU FFNs with queries. Milli et al. [105] show that when the columns of $\boldsymbol{A}$ are linearly independent, then a ReLU network is efficiently learnable with queries and their algorithm can act as $\mathcal{A}_{\mathrm{FFN}}$ to learn one-layer Transformers under the somewhat strong linear independence assumption. While learning 2-layer ReLU MLPs without any structural assumptions is quite difficult [35], Daniely and Granot [45] provide an efficient algorithm for learning ReLU FFNs with relatively much milder assumptions. While their algorithm is not directly applicable to FFNs without bias terms, we conjecture that with some modifications, their algorithm can also be applied in this scenario to learn one-layer Transformers.

5.4 Low Rank Recovery

We now consider the problem where the target matrix $\boldsymbol{W}^\star$ has a known upper bound on the rank: $\mathrm{rank}(\boldsymbol{W}^\star) \leq r$. This studies the scenario where the head dimension in an attention block is much smaller than the embedding dimension. In modern usage of Transformers, the head dimension is often small (e.g., 128) relative to the width (e.g., 4096 or 8192) [156, 28]. For query and key projection matrices $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{r \times d}$, if $r \ll d$, then $\mathrm{rank}(\boldsymbol{W}^\star) = \mathrm{rank}(\boldsymbol{K}^\top \boldsymbol{Q}) \leq r$. We show that in such a scenario, we can leverage techniques from compressed sensing [30] to recover the target parameters $\boldsymbol{W}^\star, \boldsymbol{v}^\star$ with $O(rd)$ queries, which is much more efficient than $O(d^2)$.

5.4.1 Background: Matrix Sensing

For matrices $\boldsymbol{A}, \boldsymbol{B}$, we write $\langle \boldsymbol{A}, \boldsymbol{B} \rangle = \mathsf{tr}(\boldsymbol{A}^\top \boldsymbol{B})$ for the Frobenius inner product on $\mathbb{R}^{d \times d}$, and $\|\boldsymbol{W}\|_*$ for the nuclear norm (sum of singular values). For vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$, we use $\boldsymbol{a} \otimes \boldsymbol{b} := \boldsymbol{a}\boldsymbol{b}^\top$ to denote a rank–one matrix.

Matrix sensing and the ROP model. A matrix sensing problem specifies an unknown matrix $\boldsymbol{W}^\star \in \mathbb{R}^{d_1 \times d_2}$ and a linear map

$$\mathcal{A} : \mathbb{R}^{d_1 \times d_2} \to \mathbb{R}^m, \qquad (\mathcal{A}(\boldsymbol{W}))_k = \langle \boldsymbol{A}_k, \boldsymbol{W} \rangle \quad (1 \le k \le m),$$

with known sensing matrices A_k. Cai and Zhang [30] introduced the "rank-one projection" (ROP) model that takes $\boldsymbol{A}_k = \boldsymbol{a}_k \boldsymbol{b}_k^\top$ with $\boldsymbol{a}_k \in \mathbb{R}^{d_1}$, $\boldsymbol{b}_k \in \mathbb{R}^{d_2}$ (typically i.i.d. Gaussian), so each measurement is

$$y_k = \langle \boldsymbol{a}_k \boldsymbol{b}_k^\top, \boldsymbol{W}^\star \rangle = \boldsymbol{a}_k^\top \boldsymbol{W}^\star \boldsymbol{b}_k.$$

They also introduce a concept of Restricted uniform boundedness (RUB) property [30, Definition 2.1], which is analogous to the Restricted Isometry Property (RIP) but more suited to the ROP model. They show that the RUB property leads to guaranteed recovery of low-rank matrices. In particular, given a collection of matrices $\mathcal{A}$ satisfying the RUB property and measurements $y = \mathcal{A}(\boldsymbol{W}^\star)$, one can recover a low-rank matrix $\boldsymbol{W}$ by solving a convex program which minimizes the nuclear norm $\min_{\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}} \|\boldsymbol{W}\|_*$ under the constraints that $\langle \boldsymbol{A}_k, \boldsymbol{W} \rangle = y_k$ for all k.

The final useful fact is that constructing rank-one projections by randomly sampling i.i.d. from the standard Gaussian distribution $\mathcal{N}(0, I_d)$ leads to a collection $\mathcal{A} = \{A_1, \ldots, A_m\}$ that satisfies the RUB property with probability at least $1 - e^{-\Omega(m)}$. See Section 2.1 in Cai and Zhang [30] for more details on the background described in this section.

Theorem 10 (Recovery under ROP; 30, Thm. 2.2 and Cor. 2.1)**.** *Let $\boldsymbol{A}_k = \boldsymbol{a}_k \boldsymbol{b}_k^\top$ with $\boldsymbol{a}_k \in \mathbb{R}^{d_1}$, $\boldsymbol{b}_k \in \mathbb{R}^{d_2}$ i.i.d. standard Gaussian for $k = 1, \ldots, m$ where $m = \Omega(r(d_1 + d_2))$. For some unknown $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$, the measurements are of the form $(\mathcal{A}(\boldsymbol{W})) = \langle \boldsymbol{A}_k, \boldsymbol{W} \rangle$. Then with probability at least $1 - e^{-\Omega(m)}$, the operator $\mathcal{A}$ satisfies RUB and the following convex program recovers any rank $\le r$ matrix $\boldsymbol{W}$ exactly,*

$$\min_{\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}} \|\boldsymbol{W}\|_* \quad \text{s.t.} \quad \langle \boldsymbol{A}_k, \boldsymbol{W} \rangle = y_k \quad (k = 1, \ldots, m). \tag{5.2}$$

5.4.2 Result for Low Rank

Using Theorem 10, we show that we can obtain the parameters of a single-head attention model with significantly fewer queries when the head dimension is much smaller than the width of the model.

Theorem 11. *Assume* $\boldsymbol{v}^\star \neq \mathbf{0}$ *and* $\operatorname{rank}(\boldsymbol{W}^\star) \leq r$. *Then, with access to* VQ, *there is a randomized, polynomial–time algorithm that with probability at least* $1 - e^{-\Omega(m)}$ *returns* $(\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{v}}) = (\boldsymbol{W}^\star, \boldsymbol{v}^\star)$ *using* $d + m$ *queries, where* $m = Cr(2d)$ *for an absolute constant* C.

Recovering $\boldsymbol{v}^\star$ is trivial and can be done in the same way as the vanilla problem by querying $X = [e_i^\top]$ for $i = 1, \ldots, d$.
Hence, the main focus is to recover $\boldsymbol{W}^\star$ using rank-one measurements. Fix $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$ and query

$$X = \begin{bmatrix} (\boldsymbol{a}+\boldsymbol{b})^\top \\ \boldsymbol{b}^\top \end{bmatrix}.$$

Let $s_1 = (\boldsymbol{a}+\boldsymbol{b})^\top \boldsymbol{W}^\star \boldsymbol{b}$, $s_2 = \boldsymbol{b}^\top \boldsymbol{W}^\star \boldsymbol{b}$. Then $s_1 - s_2 = \boldsymbol{a}^\top \boldsymbol{W}^\star \boldsymbol{b}$, and the attention weight on the first row equals

$$\alpha = \frac{e^{s_1}}{e^{s_1} + e^{s_2}} = \sigma(s_1 - s_2) = \sigma(\boldsymbol{a}^\top \boldsymbol{W}^\star \boldsymbol{b}).$$

Since $Xv^\star = [(\boldsymbol{a}+\boldsymbol{b})^\top \boldsymbol{v}^\star,\ \boldsymbol{b}^\top \boldsymbol{v}^\star]^\top$, the oracle returns $y = \boldsymbol{b}^\top \boldsymbol{v}^\star + \alpha\,(\boldsymbol{a}^\top \boldsymbol{v}^\star)$. If $\boldsymbol{a}^\top \boldsymbol{v}^\star \neq 0$ (this occurs almost surely for Gaussian $\boldsymbol{a}$ when $\boldsymbol{v}^\star \neq 0$), then

$$\alpha = \frac{y - \boldsymbol{b}^\top \boldsymbol{v}^\star}{\boldsymbol{a}^\top \boldsymbol{v}^\star} \in (0, 1), \qquad t := \sigma^{-1}(\alpha) = \boldsymbol{a}^\top \boldsymbol{W}^\star \boldsymbol{b} = \langle \boldsymbol{a}\boldsymbol{b}^\top,\, \boldsymbol{W}^\star \rangle.$$

Thus *one* two–row VQ produces a rank–one sensing matrix $\boldsymbol{A} := \boldsymbol{a}\boldsymbol{b}^\top$ and its linear observation $t = \langle \boldsymbol{A}, \boldsymbol{W}^\star \rangle$.
Draw i.i.d. $(\boldsymbol{a}_k, \boldsymbol{b}_k) \sim \mathcal{N}(0, I_d) \times \mathcal{N}(0, I_d)$, and for each pair query the oracle to compute

$$t_k = \sigma^{-1}\left(\frac{y_k - \boldsymbol{b}_k^\top \widehat{\boldsymbol{v}}}{\boldsymbol{a}_k^\top \widehat{\boldsymbol{v}}} \right) = \langle \boldsymbol{a}_k \boldsymbol{b}_k^\top,\, \boldsymbol{W}^\star \rangle, \qquad k = 1, \ldots, m.$$

Let $\mathcal{A}$ be the measurement operator with rows $\boldsymbol{A}_k = \boldsymbol{a}_k \boldsymbol{b}_k^\top$. We have gathered the noiseless ROP system $\mathcal{A}(\boldsymbol{W}^\star) = t \ \in \mathbb{R}^m$. Solve the convex program

$$\widehat{\boldsymbol{W}} \in \arg\min_{\boldsymbol{W} \in \mathbb{R}^{d \times d}} \|\boldsymbol{W}\|_* \quad \text{s.t.} \quad \langle \boldsymbol{a}_k \boldsymbol{b}_k^\top, \boldsymbol{W} \rangle = t_k \ (k = 1, \ldots, m).$$

By Theorem 10 (with $d_1 = d_2 = d$), if $m \geq C\, r(2d)$ then, with probability at least $1 - e^{-\Omega(m)}$ over $\{(\boldsymbol{a}_k, \boldsymbol{b}_k)\}$, the operator $\mathcal{A}$ satisfies RUB with the required condition. Hence, by Theorem 10, we have $\widehat{\boldsymbol{W}} = \boldsymbol{W}^\star$. This proves Theorem 11.
The first part uses d VQs and the recovery of $\boldsymbol{W}^\star$ uses m queries. Thus, the total is $d + m = O(rd)$. Program equation 5.2 is convex and solvable in polynomial time.

5.5 Robustness and Stability of Learning

We study a variant of the learning problem where the learner only has access to *approximate* value queries rather than exact ones. The primary motivation is that the algorithm for the noiseless case makes use of $\sigma^{-1}(\cdot)$ function which is not Lipschitz. Hence, if a teacher or an API access provider adds even a tiny amount of noise to the labels or values, then the output of the algorithm can change significantly and the guarantees do not hold. In this section, we show that, even with approximate values, with some assumptions on the norms of the parameters, we can still obtain some reasonable guarantees.
Approximate value query oracle. Fix a target function $f^\star : \mathcal{X} \to \mathbb{R}$ (here $f^\star = f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star}$). Given an input X and tolerance $\tau > 0$, the oracle returns $\mathrm{AVQ}(X; \tau)$ satisfying

$$\mathrm{AVQ}(X; \tau) \in [f^\star(X) - \tau,\ f^\star(X) + \tau],$$

and is deterministic (repeated queries on the same X return the same value). The setting is similar in spirit to the widely studied statistical query oracle [83] but is distribution-independent.
Setup assumptions. We assume $\|\boldsymbol{W}^\star\|_F \leq W$ for a known scalar $W \geq 2$, $\|\boldsymbol{v}^\star\|_2 \leq 1$, and $\|\boldsymbol{x}_i\|_2 \leq 1$. For simplicity we assume a value margin $\min_{i \in [d]} |\boldsymbol{v}_i^\star| \geq \mu > 0$. Since $\|\boldsymbol{v}^\star\|_2 \leq 1$ and $\|\boldsymbol{x}_i\|_2 \leq 1$, the output is bounded: $|f^\star(X)| \leq 1$.
Notation. Let $\sigma(t) = 1/(1 + e^{-t})$ and $\sigma^{-1}(p) = \log\big(p/(1-p)\big)$. For $\tau \in (0, 1/2)$ define $\mathrm{clip}(u; \tau, 1-\tau) = \min\{\max\{u, \tau\}, 1 - \tau\}$.
We will make use of the following elementary helper Lemma which shows that $\sigma^{-1}(\cdot)$ is Lipschitz within a certain region even if it is not globally Lipschitz.

Lemma 5. *If $\alpha^\star \in [\tau_{\mathrm{clip}},\, 1 - \tau_{\mathrm{clip}}]$ with $\tau_{\mathrm{clip}} = \sigma(-\frac{1}{2})$, then for all $\widehat{\alpha} \in \mathbb{R}$,*

$$\left|\sigma^{-1}(\mathrm{clip}(\widehat{\alpha}; \tau_{\mathrm{clip}}, 1 - \tau_{\mathrm{clip}})) - \sigma^{-1}(\alpha^\star)\right| \leq L_{\tau_{\mathrm{clip}}} \, |\widehat{\alpha} - \alpha^\star| \leq 5\, |\widehat{\alpha} - \alpha^\star|.$$

Proof. On $(0, 1)$, $(\sigma^{-1})'(p) = \frac{1}{p(1-p)}$, hence on $[\tau_{\mathrm{clip}}, 1 - \tau_{\mathrm{clip}}]$ the derivative is at most $L_{\tau_{\mathrm{clip}}} = \frac{1}{\tau_{\mathrm{clip}}(1 - \tau_{\mathrm{clip}})}$. Also, $u \mapsto \mathrm{clip}(u; \tau_{\mathrm{clip}}, 1 - \tau_{\mathrm{clip}})$ is 1-Lipschitz and maps into

$[\tau_{\text{clip}}, 1-\tau_{\text{clip}}]$. Applying the mean value theorem to σ^{-1} on this interval leads to the claim, and $L_{\tau_{\text{clip}}} \le 5$ gives the desired bound. □

Theorem 12. *Fix $d \ge 1$ and let $f^\star = f_{\boldsymbol{W}^\star, \boldsymbol{v}^\star}$ with $\|\boldsymbol{W}^\star\|_F \le W$ for a known $W \ge 2$, $\|\boldsymbol{v}^\star\|_2 \le 1$, and $\min_{i\in[d]} |\boldsymbol{v}_i^\star| \ge \mu > 0$. Assume that the learner has access to a deterministic oracle $\mathrm{AVQ}(\cdot;\tau)$ returning a value within $\pm\tau$ of $f^\star(\cdot)$. Then for any $\epsilon_v, \epsilon_W \in (0,1)$ there is a polynomial-time algorithm that makes $O(d^2)$ approximate value queries, each with tolerance $\tau = O\Big(\min\Big\{\mu,\ \frac{\epsilon_v}{\sqrt{d}},\ \frac{\mu\,\epsilon_W}{W^2 d}\Big\}\Big)$ and outputs $(\widehat{\boldsymbol{v}}, \widehat{\boldsymbol{W}})$ such that*

$$\|\widehat{\boldsymbol{v}} - \boldsymbol{v}^\star\|_2 \le \epsilon_v \qquad \text{and} \qquad \|\widehat{\boldsymbol{W}} - \boldsymbol{W}^\star\|_F \le \epsilon_W.$$

Approximating $\boldsymbol{v}^\star$ is straightforward, the key challenge is to approximate $\boldsymbol{W}^\star$. The key technical idea is to construct probes such that $\sigma^{-1}(\cdot)$ is always applied to values within certain bounds where it is Lipschitz, and consequently the error in the remaining operations can be controlled to achieve the desired guarantee.

Proof. To approximate $\boldsymbol{v}^\star$, we can simply query basis vectors, $X = [\mathbf{e}_i^\top]$ for each $i \in [d]$ with tolerance τ_v and set

$$\widehat{\boldsymbol{v}}_i := \mathrm{AVQ}([\mathbf{e}_i^\top]; \tau_v).$$

Then $|\widehat{\boldsymbol{v}}_i - \boldsymbol{v}_i^\star| \le \tau_v$ for all i, and therefore $\|\widehat{\boldsymbol{v}} - \boldsymbol{v}^\star\|_2 \le \sqrt{d}\,\tau_v$. If additionally $\tau_v \le \mu/2$, then $|\widehat{\boldsymbol{v}}_i| \ge \mu/2$ for every i, which will be used to control ratios below.
Approximating $\boldsymbol{W}^\star$. We want to estimate $\boldsymbol{W}^\star$ such that $\|\widehat{\boldsymbol{W}} - \boldsymbol{W}^\star\|_F \le \epsilon_W$. Fix $a = \frac{1}{2}$ and $b = 1/W$ so that $|ab\,\boldsymbol{W}_{ij}^\star| \le ab\|\boldsymbol{W}^\star\|_F \le \frac{1}{2}$. For a two-row input $X = \begin{bmatrix} u^\top \\ x^\top \end{bmatrix}$, we can write

$$f_{\boldsymbol{W},\boldsymbol{v}}(X) = x^\top \boldsymbol{v} + \alpha_1\,(u^\top \boldsymbol{v} - x^\top \boldsymbol{v}), \qquad \alpha_1 = \sigma\big(u^\top \boldsymbol{W} x - x^\top \boldsymbol{W} x\big). \tag{5.3}$$

To estimate entry (i,j) of $\boldsymbol{W}^\star$, use the probe

$$X = \begin{bmatrix} (b\mathbf{e}_i + a\mathbf{e}_j)^\top \\ (a\mathbf{e}_j)^\top \end{bmatrix}.$$

Plugging it to equation 5.3 we have,

$$f^\star(X) = a\,\boldsymbol{v}_j^\star + \alpha_1^\star\,(b\,\boldsymbol{v}_i^\star), \qquad \alpha_1^\star := \alpha_1(X; \boldsymbol{W}^\star) = \frac{f^\star(X) - a\,\boldsymbol{v}_j^\star}{b\,\boldsymbol{v}_i^\star}.$$

On the other hand, the logit term in equation 5.3 simplifies to $(b\mathbf{e}_i + a\mathbf{e}_j)^\top \boldsymbol{W}^\star(a\mathbf{e}_j) - (a\mathbf{e}_j)^\top \boldsymbol{W}^\star(a\mathbf{e}_j) = ab\,\boldsymbol{W}^\star_{ij}$, so

$$\alpha_1^\star = \sigma(ab\,\boldsymbol{W}^\star_{ij}) \qquad \Longrightarrow \qquad \boldsymbol{W}^\star_{ij} = \frac{1}{ab}\,\sigma^{-1}(\alpha_1^\star).$$

Since $ab\,\boldsymbol{W}^\star_{ij} \in [-\frac{1}{2}, \frac{1}{2}]$, we have $\alpha_1^\star \in [\sigma(-\frac{1}{2}),\, 1 - \sigma(-\frac{1}{2})]$. Let $\tau_{\text{clip}} := \sigma(-\frac{1}{2})$ and $L_{\tau_{\text{clip}}} := \frac{1}{\tau_{\text{clip}}(1-\tau_{\text{clip}})} \le 5$.

Estimators from approximate values. Fix (i, j) and let X be the probe above. Query the oracle once on X with tolerance τ_f and define

$$\hat{f}(X) := \mathrm{AVQ}(X; \tau_f) = f^\star(X) + \eta_f, \qquad |\eta_f| \le \tau_f.$$

Use the coordinate estimates $\widehat{\boldsymbol{v}}_i, \widehat{\boldsymbol{v}}_j$ obtained earlier with tolerance τ_v, so $\widehat{\boldsymbol{v}}_k = \boldsymbol{v}_k^\star + \eta_k$ with $|\eta_k| \le \tau_v$. Define

$$\widehat{\alpha}_1 := \frac{\hat{f}(X) - a\,\widehat{\boldsymbol{v}}_j}{b\,\widehat{\boldsymbol{v}}_i}.$$

We have

$$\widehat{\alpha}_1 - \alpha_1^\star = \frac{\hat{f}(X) - a\,\widehat{\boldsymbol{v}}_j}{b\,\widehat{\boldsymbol{v}}_i} - \alpha_1^\star = \frac{\hat{f}(X) - a\,\boldsymbol{v}_j^\star + \; a(\boldsymbol{v}_j^\star - \widehat{\boldsymbol{v}}_j) \; - \alpha_1^\star b\,\widehat{\boldsymbol{v}}_i}{b\,\widehat{\boldsymbol{v}}_i},$$

and adding and subtracting $\alpha_1^\star b\,\boldsymbol{v}_i^\star$ gives

$$\widehat{\alpha}_1 - \alpha_1^\star = \frac{\hat{f}(X) - a\,\boldsymbol{v}_j^\star - \alpha_1^\star b\,\boldsymbol{v}_i^\star \; + \; a(\boldsymbol{v}_j^\star - \widehat{\boldsymbol{v}}_j) \; + \; \alpha_1^\star b(\boldsymbol{v}_i^\star - \widehat{\boldsymbol{v}}_i)}{b\,\widehat{\boldsymbol{v}}_i}.$$

Since $f^\star(X) = a\,\boldsymbol{v}_j^\star + \alpha_1^\star b\,\boldsymbol{v}_i^\star$, the first three terms equal η_f, hence

$$|\widehat{\alpha}_1 - \alpha_1^\star| \le \left| \frac{\eta_f + a(\boldsymbol{v}_j^\star - \widehat{\boldsymbol{v}}_j) + \alpha_1^\star b(\boldsymbol{v}_i^\star - \widehat{\boldsymbol{v}}_i)}{b\,\widehat{\boldsymbol{v}}_i} \right| \le \frac{|\eta_f| + a\,|\widehat{\boldsymbol{v}}_j - \boldsymbol{v}_j^\star|}{|b\,\widehat{\boldsymbol{v}}_i|} + \frac{|\widehat{\boldsymbol{v}}_i - \boldsymbol{v}_i^\star|}{|\widehat{\boldsymbol{v}}_i|}. \quad (5.4)$$

From tolerances to accuracy. Assume $\tau_v \le \mu/2$, so $|\widehat{\boldsymbol{v}}_i| \ge \mu/2$ for all i. Then

$$\frac{1}{|b\,\widehat{\boldsymbol{v}}_i|} \le \frac{2}{b\mu} = \frac{2W}{\mu}, \qquad \frac{|\widehat{\boldsymbol{v}}_i - \boldsymbol{v}_i^\star|}{|\widehat{\boldsymbol{v}}_i|} \le \frac{2}{\mu}\,\tau_v.$$

Using $|\eta_f| \le \tau_f$ and $|\widehat{\boldsymbol{v}}_j - \boldsymbol{v}_j^\star| \le \tau_v$ in equation 5.4 yields

$$|\widehat{\alpha}_1 - \alpha_1^\star| \le \frac{2W}{\mu}\big(\tau_f + a\,\tau_v\big) + \frac{2}{\mu}\tau_v = \frac{2W}{\mu}\tau_f + \frac{W+2}{\mu}\tau_v.$$

With the choices

$$\tau_f \le \frac{\mu\,\epsilon_W}{80\,W^2 d}, \qquad \tau_v \le \min\left\{\frac{\mu}{2},\; \frac{\mu\,\epsilon_W}{80\,W^2 d}\right\},$$

we obtain $|\widehat{\alpha}_1 - \alpha_1^\star| \le \epsilon_W/(10Wd)$. Define

$$\widehat{\boldsymbol{W}}_{ij} := \frac{1}{ab}\,\sigma^{-1}\big(\mathrm{clip}(\widehat{\alpha}_1; \tau_{\mathrm{clip}}, 1-\tau_{\mathrm{clip}})\big).$$

By Lemma 5,

$$|\widehat{\boldsymbol{W}}_{ij} - \boldsymbol{W}^\star_{ij}| \le \frac{L_{\tau_{\mathrm{clip}}}}{ab}\,|\widehat{\alpha}_1 - \alpha_1^\star| \le \frac{\epsilon_W}{d} \quad \text{for all } (i,j),$$

and hence $\|\widehat{\boldsymbol{W}} - \boldsymbol{W}^\star\|_F \le \epsilon_W$.

Query complexity and required precision. The reconstruction uses d one-row queries and d^2 two-row probe queries, for a total of $d+d^2$ queries. The only requirement is that the oracle can answer these queries with additive error at most $\tau_v, \tau_f = O\big(\frac{\mu\,\epsilon_W}{W^2 d}\big)$ (up to constants), and $\tau_v \le \mu/2$ to ensure the denominator in equation 5.4 is bounded away from 0. □

5.6 Learnability of Multi-head Attention with Queries

In this section, we consider the problem of learning a multi-head attention layer. Let d_h denote the embedding dimension of each head, and d the total embedding dimension, such that the number of heads is $H = d/d_h$. The parameter matrices for the multi-head attention are given by $\boldsymbol{W}^{(1)}, \dots, \boldsymbol{W}^{(H)} \in \mathbb{R}^{d\times d}$, and the corresponding projection vectors by $\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(H)} \in \mathbb{R}^d$. Let W and v denote the sets of all weight matrices and vectors, respectively. The output of the multi-head attention layer is then defined as

$$f^H_{W,v}(X) = \sum_{h=1}^{H} \boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)})^\top (X\boldsymbol{v}^{(h)}).$$

where $\boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)})$ represents the attention weights computed for head h.

We observe that, unlike the single-head attention layer, not every set of parameters (W, v) corresponds to a unique function $f^H_{W,v}(X)$. In particular, we show the following proposition.

Proposition 13. *Let $f^H_{W,v}$ be a multi-head attention layer with H heads and with a set of parameters (W, v). Then, there exist multiple set of parameters (W, v) that lead to the same function $f^H_{W,v} : \mathbb{R}^{N\times d} \to \mathbb{R}$.*

Proof. We consider the set of weight parameters W such that the weight matrices of all heads are identical, i.e., $\boldsymbol{W}^{(h)} = \boldsymbol{A}$ for all $h \in [H]$, where $\boldsymbol{A}$ is any fixed matrix. Let $\boldsymbol{b} \in \mathbb{R}^d$ be any arbitrary non zero vector and let $\lambda_1, \dots, \lambda_H$ be H non negative

weights such that $\sum_{i=1}^{H} \lambda_i = 1$. For any such weights, set $\boldsymbol{v}^{(h)} = \lambda_h \boldsymbol{b}$ for all $h \in [H]$. Then, the multi-head attention model computes,

$$f^H_{W,v}(X) = \sum_{h=1}^{H} \boldsymbol{\alpha}^{(h)}(X, \boldsymbol{W}^{(h)})^\top (X\boldsymbol{v}^{(h)}) = \boldsymbol{\alpha}(X, \boldsymbol{A})^\top \sum_{h=1}^{H} \lambda_h (X\boldsymbol{b}) = \boldsymbol{\alpha}(X, \boldsymbol{A})^\top (X\boldsymbol{b}).$$

Thus, for different weights $\lambda_1, \ldots, \lambda_H$, we will have different projection vectors $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(H)}$ but all of them would lead to the same function as defined above. □

Discussion. As a consequence of Proposition 13, the parameters of the multi-head attention layer cannot be uniquely identified from value queries alone, and achieving guarantees like Theorem 9 is not possible. For exact learning with queries to be feasible, the parameter-function map needs to be one-to-one. Different from efficient learning, we say a class of functions is identifiable from queries if there exists some set of input-output pairs that uniquely determines the target function.

Learnability of multi-head attention with queries is left for future work, and many questions around that remain open. We discuss some potential directions and approaches to tackle the multi-head problem. (i) It could be useful to first identify structural assumptions under which the class is identifiable. We observe that if the parameters of each head $(\boldsymbol{W}^{(h)}, \boldsymbol{v}^{(h)})$ lie in mutually orthogonal subspaces, then the class is identifiable (up to permutations). It could also be feasible to efficiently learn the parameters by decomposing the Hessian with respect to probe vector $\boldsymbol{z}$ for inputs $X = \begin{bmatrix} (\boldsymbol{z} + \mathbf{e}_j)^\top \\ \mathbf{e}_j^\top \end{bmatrix}$ and using block-diagonalisation methods [109] to identify the basis of each subspace. (ii) Apart from parameter estimation, an interesting direction is to understand functional equivalence. Given access to queries, is it possible to learn a function that closely approximates the target even if the parameters are different? Even basic decidability questions around this are not clearly understood – for instance, given two sets of parameters of multi-head attention models, is the problem of checking whether they represent the same function decidable? (iii) Further, it could be interesting to analyse whether distribution-dependent PAC guarantees (such as Chen et al. [35] for MLPs) are attainable, unlike distribution-independent ones in our work.

Chapter 6

Learning Automata with Queries and Support of Language Models

6.1 Introduction

Language models (LMs) are now deployed across text, vision, and bioinformatics; yet their internal computation and potential outputs they generate remain difficult to interpret. This motivates a basic question: given black-box access to a model, can we extract a compact, interpretable formal object, such as a deterministic finite automaton (DFA), that accepts (approximately) the same strings as those that lie in the model's generative support? We develop a formal framework for this problem and study its learnability in a setting that closely mirrors how LMs are typically used in practice.

We formalise and study the problem of learnability of languages in the *Next Symbol Prediction* (NSP) setting. Here, a learner receives only *positive* strings from a target language, together with rich supervision for every prefix: a membership bit indicating whether the prefix itself is in the language and a vector of "continuation" bits indicating which next symbols admit some accepting continuation. A prediction is correct only if the hypothesis matches *all* membership and continuation labels at *every* prefix of the example.

This setup has a natural interpretation in the context of language models: when decoding with top-p [74], top-k, or min-p [108] sampling, the per-prefix continuation set is precisely the set of admissible next tokens, and termination corresponds to allowing the end-of-sequence token. In particular, positive-only NSP supervision is naturally obtained from black-box models and avoids the need for inventing artificial distributions over negative strings; at the same time, the requirement for correct predictions at every prefix makes the task challenging. Figure 6.1 illustrates NSP

labels on a Dyck example and how admissible-next-token sets can be read from a language model.

Additionally, while NSP has been widely used to *evaluate* sequence models on formal-language benchmarks (see e.g. Gers and Schmidhuber [59], Suzgun et al. [150], Ebrahimi et al. [51], Bhattamishra et al. [18] and references therein), the learnability of languages under NSP and its relationship to conventional binary classification has not been established.

Our contributions. We investigate the question of learnability in the NSP setting within the computational learning theory framework by studying computational complexity and oracle requirements. We also conduct a systematic empirical evaluation with several regular languages, using Transformer-based language models as teachers. This chapter makes the following contributions.

(i) Identifiability and hardness. We give a PAC-style formulation of learning using NSP labels and show that positive examples augmented with NSP labels are information-theoretically sufficient to identify minimal DFAs. Concretely, distinct minimal DFAs always disagree on the NSP labelling of some positive string, yielding finite teaching sets and a well-defined equivalence oracle in the NSP model. At the same time, we prove that NSP supervision does *not* remove the key *computational barriers* for learning regular languages. The key technical argument is a construction that renders all but one continuation label uninformative, allowing a reduction to well-known hardness results [84]. Thus, even with the richer labels, efficient (improper) learning remains computationally intractable (under standard cryptographic assumptions). We further show that identification with membership queries alone cannot be achieved in polynomial time for certain natural DFA families, even when each query returns all NSP labels. Together, these results suggest that while NSP labels offer some benefit, they do not, in general, circumvent computational hardness.

(ii) Learning with a language-model teacher. Motivated by the hardness results, we study a more powerful, though still practically motivated, learning framework based on prefix-conditional generative queries. These can easily be simulated using black-box access to an LM. In addition to membership queries, the learner can issue *generative* queries that return positively labelled NSP strings conditioned on a *prefix* prompt. We extend Angluin's $L^\star$ algorithm [7] to design a new algorithm we denote $L^\star_{\text{nsp}}$, that uses membership and generative queries to construct a DFA consistent with the observed NSP labels. The guarantee is distribution-specific: $L^\star_{\text{nsp}}$ PAC-learns with respect to the distribution induced by the teacher language model. This perspective is aligned with our goal of identifying the model's (truncated) support and is particularly

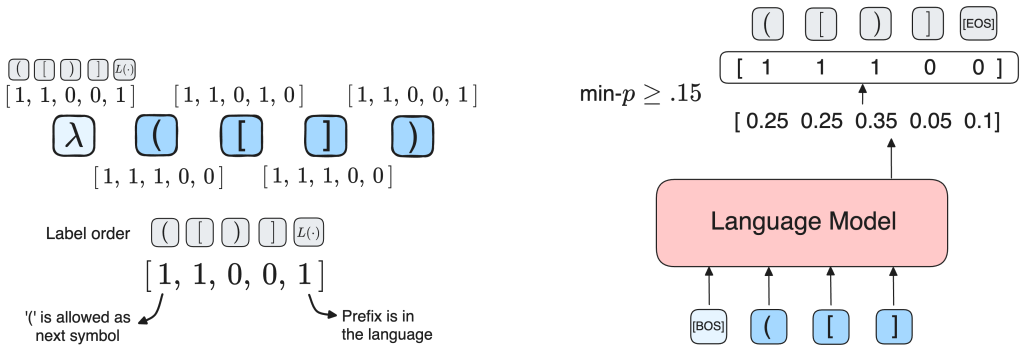


Figure 6.1: *Left:* An example of a string with NSP labels from the Dyck-2 language. The language consists of well-balanced parentheses with two types of brackets. *Right:* An illustration of how NSP labels can be obtained from a language model. See Section 6.2 for details.

appealing in the context of generative models, where the target distribution over negative strings is either undefined or arbitrary. Conditional generation is both a natural capability of modern LMs and turns out to be a powerful query primitive for efficient learning in the NSP framework.

(iii) Empirical evaluation and analysis. We apply the $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm to extract DFAs from Transformer teachers trained on eleven regular languages of varying complexity, including the Tomita grammars, Parity, and bounded Dyck languages. We study how NSP accuracy, the number of extracted states, and runtime scale with the number of positive training strings. Across tasks, a modest amount of positive data with NSP labels typically suffices to recover the target automata or their teacher-support counterparts. When the teacher is imperfect (e.g., for Parity, Tomita-5), the extracted DFA reveals systematic errors by identifying strings in the symmetric difference between the target and the teacher's support. Ablations further indicate that the continuation labels are heavily used on languages with transitions to dead states (e.g., bounded Dyck), leading to improved sample complexity over binary labels alone. These experiments underscore a practical point: while NSP labels cannot break worst-case computational barriers, they are easy to obtain from modern sequence models and can be leveraged effectively in practice.

6.2 Problem Definition

Notation. A deterministic finite automaton (DFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F_A)$ with finite state set Q, alphabet Σ, transition function $\delta : Q \times \Sigma \to Q$, start state q_0, and a subset of accepting states $F_A \subseteq Q$. The language of A is $L_A \subseteq \Sigma^*$; write

$A(x) = L_A(x) \in \{0,1\}$ and $|A|$ denotes the number of states $|Q|$. Fix an order $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$. For $x = w_1 \cdots w_N$, let the length-n prefix be $x_{:n} := w_1 \cdots w_n$ for $0 \le n \le N$. We use q_{dead} to denote a dead state with $q_{\text{dead}} \notin F_A$ and $\delta(q_{\text{dead}}, \sigma) = q_{\text{dead}}$ for all $\sigma \in \Sigma$ (unique, if present, in a minimal DFA). We use DFA_n to denote the class of DFAs with at most n states.

6.2.1 Next Symbol Prediction (NSP) Setting

For a language $L \subseteq \Sigma^*$ and any $x \in \Sigma^*, \sigma \in \Sigma$, define the *continuation bit*

$$\varphi_L(x, \sigma) \;:=\; \mathbb{I}\big[\, \exists s \in \Sigma^* \text{ s.t. } x \cdot \sigma \cdot s \in L \,\big].$$

If L is regular with minimal DFA A, then with $q = \delta(q_0, x)$ we have $\varphi_A(x, \sigma) = 0$ if and only if $\delta(q, \sigma) = q_{\text{dead}}$. The continuation vector at q is defined as $\varphi_A(q) = [\varphi_A(q, \sigma_1), \ldots, \varphi_A(q, \sigma_{|\Sigma|})] \in \{0,1\}^{|\Sigma|}$. For strings $x \in \Sigma^*$, write $\varphi_A(x) := \varphi_A(\delta(q_0, x))$ and $\varphi_A(x, \sigma) := \varphi_A(\delta(q_0, x), \sigma)$.
A positive NSP–labeled example is a string $x = w_1 \cdots w_N \in L$ together with, for every prefix $x_{:n}$ $(0 \le n \le N)$, its membership $L(x_{:n})$ and all continuation bits $(\varphi(x_{:n}, \sigma))_{\sigma \in \Sigma}$. We collect these as

$$f_L(x) \;:=\; \Big((\varphi_L(x_{:n}, \sigma_i))_{i=1}^{|\Sigma|},\, L(x_{:n}) \Big)_{n=0}^{N} \in \{0,1\}^{(|\Sigma|+1)(N+1)}.$$

We will use the term NSP labels to refer to such labels (See Fig. 6.1, left for an illustration). We instantiate predictors via automata. For a DFA A define $f_A(x) := \big((\varphi_A(x_{:n}, \sigma_i))_{i=1}^{|\Sigma|},\, L_A(x_{:n}) \big)_{n=0}^{|x|}$. Let $f_{A^\star}$ denote the target NSP labelling function. The per-example loss is the 0/1 sup-norm mismatch $\text{err}(f_A(x), f_{A^\star}(x)) := \|f_A(x) - f_{A^\star}(x)\|_\infty = \mathbb{I}[f_A(x) \neq f_{A^\star}(x)]$, i.e., it equals 1 iff *any* membership/continuation label for any prefix is wrong; the NSP loss on $\mathcal{D}$ (supported on strings in $L_{A^\star}$) is $\mathcal{L}_{\text{NSP}}(f_A; f_{A^\star}, \mathcal{D}) := \mathrm{E}_{x \sim \mathcal{D}}[\text{err}(f_A(x), f_{A^\star}(x))]$. Because all $(|\Sigma|+1)(|x|+1)$ labels must be simultaneously correct, NSP is stringent in the sense that a naive random guesser has a near-zero chance of zero error on a typical example (contrast with $\approx$50% in binary classification).

6.2.2 Learning the Truncated Support of Language Models

For a vocabulary Σ containing [EOS], let a language model LM define next-token probabilities $p_{\text{LM}}(\cdot \mid y)$ on Σ for each prefix $y \in \Sigma^*$. A sampling/truncation rule $\mathcal{T}$ (e.g., top-p, top-k, per-step min-p) maps y to the admissible next-symbol set $\mathcal{C}_{\mathcal{T}}(y) \subseteq \Sigma$, with [EOS] $\in \mathcal{C}_{\mathcal{T}}(y)$ exactly when y may terminate (see Fig. 6.1, right for

an illustration). The $\mathcal{T}$-*truncated support* of LM is the set of all strings that can be generated by running LM under $\mathcal{T}$. Formally,

$$L_{\mathrm{LM}}^{\mathcal{T}} := \Big\{ x = w_1 \cdots w_N : \forall\, 0 \le n < N,\ w_{n+1} \in \mathcal{C}_{\mathcal{T}}(w_1 \cdots w_n) \text{ and } [\mathsf{EOS}] \in \mathcal{C}_{\mathcal{T}}(x) \Big\}.$$

The NSP labelling oracle induced by the language model LM and truncation strategy $\mathcal{T}$ is then,

$$L_{\mathrm{LM}}^{\mathcal{T}}(y) := \mathbb{I}[[\mathsf{EOS}] \in \mathcal{C}_{\mathcal{T}}(y)], \qquad \varphi^{\mathcal{T}}(y, \sigma) := \mathbb{I}[\sigma \in \mathcal{C}_{\mathcal{T}}(y)] \ \ (\sigma \in \Sigma),$$

and we set $f_{\mathrm{LM}}^{\mathcal{T}}(x) := \big((\varphi^{\mathcal{T}}(x_{:n}, \sigma_i))_{i=1}^{|\Sigma|},\ L_{\mathrm{LM}}^{\mathcal{T}}(x_{:n})\big)_{n=0}^{|x|}$. If generation under $\mathcal{T}$ terminates almost surely, then for any admissible step there exists a finite accepting continuation, so $\varphi^{\mathcal{T}}$ coincides with the global NSP semantics above.

From NSP learning to learning support. Let $\mathcal{D}_{\mathrm{LM}}^{\mathcal{T}}$ be the distribution of strings generated by LM under $\mathcal{T}$. Any PAC learner that, from NSP-labeled positive examples $(x, f^\star(x))$, outputs $\hat{f}$ with $\mathcal{L}_{\mathrm{NSP}}(\hat{f}; f^\star, \mathcal{D}) \le \epsilon$ immediately yields, by instantiating $f^\star = f_{\mathrm{LM}}^{\mathcal{T}}$ and $\mathcal{D} = \mathcal{D}_{\mathrm{LM}}^{\mathcal{T}}$, a procedure that learns the $\mathcal{T}$-truncated support of LM with NSP error at most ϵ.

Oracle simulation. With black-box access to LM and rule $\mathcal{T}$, one can simulate the typical example oracle $\mathrm{EX}(f_{\mathrm{LM}}^{\mathcal{T}}; \mathcal{D}_{\mathrm{LM}}^{\mathcal{T}})$ by sampling $x \sim \mathcal{D}_{\mathrm{LM}}^{\mathcal{T}}$ and returning $(x, f_{\mathrm{LM}}^{\mathcal{T}}(x))$. Membership queries can be computed by checking if a string takes an admissible path based on the truncation strategy $\mathcal{T}$ and if $[\mathsf{EOS}]$ is permissible at the last step.

6.3 Preliminaries

We define Probabilistic DFAs (PDFAs) formally here. Our result on learning with membership and generative queries (Theorem 19) has direct implications on learning the support of PDFAs. We also use PDFAs to generate strings for training Transformer language models.

Probabilistic DFAs (PDFAs). A probabilistic DFA is a DFA equipped with a stochastic emission rule. Formally, a PDFA is a tuple

$$\mathcal{P} = (Q, \Sigma, \delta, q_0, F, \pi),$$

where $(Q, \Sigma, \delta, q_0, F)$ is a DFA and, for each $q \in Q$, $\pi(\cdot \mid q)$ is a probability distribution on $\Sigma \cup \{\texttt{[EOS]}\}$ where `[EOS]` denotes the termination symbol. At state q, the generator samples $w \in \Sigma \cup \{\texttt{[EOS]}\}$ according to $\pi(\cdot \mid q)$; if $w \in \Sigma$ the next state is $\delta(q, w)$, and

if $w = \texttt{[EOS]}$ the sequence terminates. We say a string $x = w_1 \cdots w_N \in \Sigma^*$ is in the *support* of $\mathcal{P}$ if

$$\pi(w_1 \mid q_0) \cdot \pi(w_2 \mid \delta(q_0, w_1)) \cdots \pi(w_N \mid \delta(q_0, x_{:N-1}))\pi(\texttt{[EOS]} \mid \delta(q_0, x)) > 0.$$

The DFA that accepts exactly this support is the *support DFA* of $\mathcal{P}$.

NSP labels from PDFAs. In a PDFA $\mathcal{P}$, the NSP labels coincide with positivity of the local emission probabilities:

$$\varphi(y, \sigma) = \mathbb{I}\{\pi(\sigma \mid \delta(q_0, y)) > 0\}, \qquad L(y) = \mathbb{I}\{\pi(\texttt{[EOS]} \mid \delta(q_0, y)) > 0\}.$$

Thus, the NSP labelling oracle exposes exactly the admissible next symbols and termination at each prefix; for a PDFA this recovers the (untruncated) support of $\mathcal{P}$. The following is an elementary but fundamental fact about minimal DFAs due to the Myhill-Nerode Theorem that is at the heart of many of our proofs and constructions.

Lemma 6 (DFA basic fact)**.** *Let A be a DFA recognising L_A. For any two distinct strings $x, y \in \Sigma^*$, if there exists a suffix $s \in \Sigma^*$ such that $L_A(x \cdot s) \neq L_A(y \cdot s)$ then the strings x and y lead to two distinct states in the DFA A, i.e., $\delta_A(q_0, x) \neq \delta_A(q_0, y)$. If A is minimal and no such suffix exists, then they lead to the same state $\delta_A(q_0, x) = \delta_A(q_0, y)$.*

In a minimal DFA, distinct states are pairwise distinguishable by some continuation; conversely, strings with indistinguishable residual languages must reach the same state. This is the Myhill–Nerode characterization of minimality.

Lemma 7. *Let the dead state q_{dead} be a state such that $q_{\text{dead}} \notin F$ and $\delta(q_{\text{dead}}, \sigma) = q_{\text{dead}}$ for all $\sigma \in \Sigma$. Then, a minimal DFA has at most one dead state.*

Assume there are two dead states. Since every suffix from a dead state is rejected, there cannot be a suffix that is accepted by one state and rejected by another. Further, by definition, both of them are non-final. Hence, they are Myhill–Nerode equivalent and must be merged in a minimal DFA.

Lemma 8. *Let DFA_n be the class of DFAs over a fixed alphabet Σ with at most n states. Then $\log |\text{DFA}_n| = \mathcal{O}(n \log n)$.*

This is a well-known result [47]. For a DFA with at most n states, the transition function is a map $\delta : Q \times \Sigma \to Q$, so $|Q \times \Sigma| \leq n|\Sigma|$. Thus, there are at most $n^{|\Sigma| \cdot n}$ possible transition functions. Moreover, there are at most 2^n choices for the set of accepting states and at most n choices for the initial state. Hence, the total number of possible DFAs is at most $2^n \cdot n^{|\Sigma| \cdot n + 1}$. Thus, $\log |\text{DFA}_n| = \mathcal{O}(n \log n)$.

6.4 Identifiability and Equivalence in the NSP setting

Before studying efficient learnability, it is first necessary to establish whether NSP labels are *necessary* and *sufficient*, in an information-theoretic sense, to identify a target language from positive examples alone. By *unique identification from positive* NSP *data* for a target DFA $A^\star$ with $L_{A^\star} \neq \emptyset$, we mean that there exists a finite set $S \subseteq L_{A^\star}$ such that

$$\forall A \in \mathrm{DFA}_n, \qquad \Big(\forall x \in S,\ f_A(x) = f_{A^\star}(x) \Big) \implies A \equiv A^\star. \tag{6.1}$$

Any such S will be called a (positive) NSP *teaching set* for $A^\star$.

Equivalence query oracle. A closely related question is whether an equivalence query oracle is well-defined in this setting. Let $\hat{A}$ and $A^\star$ be a hypothesis and target DFA, respectively. In the NSP setting, a valid Equivalence Query oracle $\mathrm{EQ}^+_{\mathrm{nsp}}(\hat{A}; A^\star)$ with respect to target $A^\star$ should take a hypothesis $\hat{A}$ as input and output 'equivalent' if $L_{\hat{A}} = L_{A^\star}$ or else it should return a counterexample $x \in L_A^\star$ such that $f_{\hat{A}}(x) \neq f_{A^\star}(x)$. In other words, the counterexample is a string that is accepted by the target DFA $A^\star$ but disagrees with $\hat{A}$ on at least one of the NSP labels.

Note that positive strings alone without additional labels are not sufficient for such identifiability: over $\Sigma = \{0, 1\}$, let $L_A = \Sigma^*$ and $L_{A^\star} = 1^*$. Every positive example $x \in 1^*$ is accepted by both, so no positive counterexample exists. The same obstruction persists even if, in addition, the oracle reveals the *membership of each prefix*: for any $x \in 1^*$ and any prefix y of x we have $L_A(y) = L_{A^\star}(y) = 1$, so positive examples with prefix-membership labels still cannot distinguish A from $A^\star$. The key property of the NSP labels is that the continuation bits convey information about strings *not* in the language: $\varphi(y, \sigma) = 0$ certifies that no extension of $y\sigma$ is accepted. This additional information suffices to separate distinct minimal DFAs using positive examples only.

Proposition 14. *Let $A \neq A^\star$ be minimal DFAs with $L_{A^\star} \neq \emptyset$. Then there exists $x \in L_{A^\star}$ such that $f_A(x) \neq f_{A^\star}(x)$. Equivalently, the oracle $\mathrm{EQ}(A; A^\star)$ is well-defined: it either returns "equivalent" or a positive counterexample $(x, f_{A^\star}(x))$.*

Proof. Since $A \neq A^\star$, there exists a string $z \in \Sigma^*$ with $A(z) \neq A^\star(z)$. If $A^\star(z) = 1$, we may take $x = z$; the NSP vectors then disagree in the membership coordinate for the full prefix z, so $f_A(x) \neq f_{A^\star}(x)$ and we are done. Thus assume

$$A(z) = 1 \quad \text{and} \quad A^\star(z) = 0,$$

and let $q' = \delta_{A^\star}(q_0, z)$ be the state that the DFA $A^\star$ reaches after traversing the string z. We distinguish two possibilities for q'.

Case (i): $q' \neq q_{\text{dead}}$**.** Because $A^\star$ is minimal, any non-accepting state distinct from the dead state has an accepting continuation (otherwise it would be equivalent to q_{dead} and hence identified with it by minimality). Hence there exists a suffix $s \in \Sigma^*$ such that $A^\star(z \cdot s) = 1$.
Let $x := z \cdot s$. In the NSP label sequence for x, the membership coordinate at the prefix z is

$$L_A(z) = 1 \quad \text{and} \quad L_{A^\star}(z) = 0,$$

so $f_A(x) \neq f_{A^\star}(x)$. The oracle may therefore return this positive example x together with $f_{A^\star}(x)$.
Case (ii): $q' = q_{\text{dead}}$**.** Write $z = w_1 \cdots w_N$ with $w_i \in \Sigma$ and set $z_{:i} := w_1 \cdots w_i$ for $0 \leq i \leq N$ (with $z_{:0} = \lambda$). There exists an index $i \in \{0, \ldots, N-1\}$ such that

$$\delta_{A^\star}(q_0, z_{:i}) \neq q_{\text{dead}} \qquad \text{and} \qquad \delta_{A^\star}(q_0, z_{:i+1}) = q_{\text{dead}}. \tag{6.2}$$

$A^\star(z) = 0$ and the DFA $A^\star$ ends at q_{dead} after traversing z, so select the first position at which q_{dead} is entered. By minimality (as in Case (i)), the non-dead state $\delta_{A^\star}(q_0, z_{:i})$ admits an accepting continuation; hence there exists a suffix $s \in \Sigma^*$ with $A^\star(z_{:i} \cdot s) = 1$. For the continuation bit at the prefix $z_{:i}$ and the next symbol w_{i+1}, equation 6.2 implies

$$\varphi_{A^\star}(z_{:i}, w_{i+1}) = 0.$$

On the other hand, since $A(z) = 1$ and $z = z_{:i}\, w_{i+1}\, (w_{i+2} \cdots w_N)$, there exists a suffix (namely $w_{i+2} \cdots w_N$) witnessing that

$$\varphi_A(z_{:i}, w_{i+1}) = 1.$$

Now set $x := z_{:i} \cdot s$. This x is accepted by $A^\star$, and the two NSP labelings disagree at the continuation coordinate $(z_{:i}, w_{i+1})$: $f_A(x) \neq f_{A^\star}(x)$.
Thus x is a valid counterexample for the oracle.
In either case, there exists a positive example x (accepted by $A^\star$) with $f_A(x) \neq f_{A^\star}(x)$ and if no such counterexample exists, then $A = A^\star$. □

Consequences. Proposition 14 has two immediate consequences. First, finite teaching sets exist: for each $A \in \text{DFA}_n \backslash \{A^\star\}$, choose a witness $x_A \in L_{A^\star}$ with $f_A(x_A) \neq f_{A^\star}(x_A)$ guaranteed by the proposition, and set $S := \{x_A : A \in \text{DFA}_n \setminus \{A^\star\}\} \subseteq L_{A^\star}$. Then S is a positive NSP teaching set for $A^\star$ in the sense of equation 6.1. Second, it also implies that the equivalence query oracle is well-defined in the NSP setting, which is crucial for exact learning to be feasible.

6.5 Hardness of Learning from Random Examples

We study efficient PAC learnability in the NSP setting. An algorithm $\mathcal{A}$ is an efficient PAC learner for a class $\mathcal{F}$ if for every $f \in \mathcal{F}$ and every distribution $\mathcal{D}$ supported on positive examples, $\mathcal{A}$ runs in polynomial time on NSP–labeled inputs and outputs $\hat{f}$ such that, with probability at least $1-\delta$, $\mathcal{L}_{\mathrm{NSP}}(\hat{f}; f, \mathcal{D}) \leq \epsilon$.

Note that the continuation labels can be highly informative for certain classes. Consider *Conjunctions*: Boolean monomials $f : \{0,1\}^N \to \{0,1\}$, e.g., $f(z) = z_2 \wedge \bar{z}_4$. In the conventional classification model, learning Conjunctions is known to require $\Theta(N)$ labeled examples. In the NSP model, one positive example $x \in f^{-1}(1)$ suffices. For each prefix $x_{:k}$, the label $\varphi(x_{:k}, 0)$ equals 0 if and only if the literal z_{k+1} appears in the target; likewise, $\varphi(x_{:k}, 1) = 0$ if and only if the literal $\bar{z}_{k+1}$ appears. Thus, by reading the continuation labels across the N prefixes, the learner recovers exactly which literals are present and hence identifies the target monomial from a single positive NSP example. While the NSP labels may provide a statistical advantage for some classes, we show that in the general case, they are not enough to remove the computational barriers for learning DFAs.

We relate learnability in the NSP setting to standard PAC learning for acyclic DFAs that accept only strings of a fixed length. Throughout this section, we work over the binary alphabet $\Sigma = \{0,1\}$.

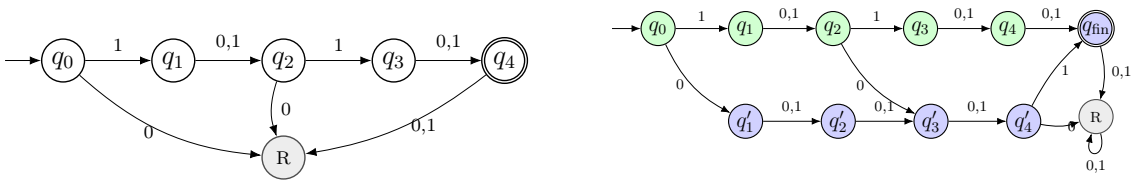
Notation for classes. For $N \in \mathbb{N}$ and a polynomial $p(\cdot)$, define

$$\mathrm{ADFA}^N_{p(N)} \;:=\; \big\{A :\; A \text{ is a DFA with at most } p(N) \text{ states and } L_A \subseteq \{0,1\}^N\big\},$$

and write $\mathrm{ADFA}_{p(\cdot)} := \bigcup_{N \geq 1} \mathrm{ADFA}^N_{p(N)}$. As defined earlier, DFA_n denotes the class of DFAs with at most n states. Under the convention that the transition function is total, any minimal $A \in \mathrm{ADFA}^N_{p(N)}$ has a dead (sink) state with self-loops on both input symbols. If instead one allows a partial transition function δ, interpreting undefined transitions as implicit moves to a dead state, then the automata in $\mathrm{ADFA}^N_{p(N)}$ are acyclic in the usual sense.

Background. Kearns and Valiant [84] show that, for a suitable polynomial p, weak PAC learning of $\mathrm{ADFA}_{p(\cdot)}$ in the conventional classification (binary-label) model is as hard as inverting certain cryptographic functions.

Theorem 15 (Kearns and Valiant [84])**.** *There exists a polynomial $p(\cdot)$ such that the problems of inverting RSA, , recognising quadratic residues, and factoring Blum integers, are probabilistic polynomial-time reducible to weakly learning $\mathrm{ADFA}_{p(\cdot)}$ in the standard PAC setting.*



(a) DFA for $z_1 \wedge z_3$, $x \in \{0,1\}^4$.

(b) Transformed DFA $(z_1 \wedge z_3) \vee z_5$, $x \in \{0,1\}^5$.

Figure 6.2: Transformation from Section 6.5. In (b), green states are from the original DFA; blue states $q'_1, \ldots, q'_4$ ensure every prefix of length $< N$ has continuation labels $[1,1,0]$, and q'_4 routes to accept on input 1 (to the dead state on 0).

We prove that an efficient PAC algorithm for $\text{ADFA}^N_{p(N)}$ in the NSP setting would yield an efficient PAC algorithm for $\text{ADFA}^N_{p(N)}$ in the conventional classification setting. Together with Theorem 15, this implies cryptographic hardness for NSP learning of these Boolean Acyclic DFAs and consequently the general class of DFAs.
Fix $N \geq 1$ and a target DFA $A = (Q, \Sigma, \delta, q_0, F)$ with $L_A \subseteq \{0,1\}^N$. We first record a basic structural property of *minimal* DFAs for fixed-length languages.

Lemma 9 (Unique depth in minimal $\text{ADFA}^N_{p(N)}$)**.** *If A is minimal and $L_A \subseteq \{0,1\}^N$, then every state $q \in Q \setminus \{q_{\text{dead}}\}$ is reachable by strings of exactly one length $\ell(q) \in \{0,1,\ldots,N\}$. Consequently, every transition increases depth by one: if $\delta(q,\sigma) = q'$ with $q \neq q_{\text{dead}}$ and $q' \neq q_{\text{dead}}$, then $\ell(q') = \ell(q) + 1$. Acceptance occurs only at depth N.*

Proof. Let $q \in Q \setminus \{q_{\text{dead}}\}$ be reachable both by a string of length ℓ and by a string of length ℓ' with $\ell < \ell'$. The residual language at q is $R(q) := \{\, s \in \Sigma^* : \delta(q,s) \in F \,\}$. If q is reached after ℓ symbols, then every $s \in R(q)$ must have length exactly $N - \ell$; if q is reached after ℓ' symbols, then every $s \in R(q)$ must have length exactly $N - \ell'$. Since $N - \ell \neq N - \ell'$, these sets are disjoint; hence $R(q)$ must be empty, contradicting $q \neq q_{\text{dead}}$ in a minimal DFA. Thus, each non-dead state has a unique depth $\ell(q)$. Any transition consumes one symbol, so $\ell(q') \leq \ell(q) + 1$; equality must hold by uniqueness of depth. Finally, if $q \in F$ had $\ell(q) \neq N$, then L_A would contain strings of length other than N, contrary to the assumption. □

By Lemma 9, we may index non-dead states by their unique depth $\ell(q) \in \{0,\ldots,N\}$ (the dead state has no depth).
Construction. The key technical idea is the construction in the following Lemma, where we construct from any $A \in \text{ADFA}^N_{p(N)}$ a DFA $A^{\oplus}$ over length-$N{+}1$ inputs whose NSP labels are uninformative before length N, while at length N the continuation

bit for symbol 0 recovers A's label. More precisely, we will construct a DFA $A^{\oplus}$ with at most $|A| + N + 1$ states such that: (i) the NSP labels of any prefix of length $< N$ will be $(1, 1, 0)$, and (ii) crucially, the NSP label for a string $u \in \Sigma^N$ of length N will be $(A(u), 1, 0)$. The ADFA $A^{\oplus}$ accepts all strings x of length $N + 1$ ending with 1 and some ending with 0 depending on $A(x_{:N})$. Thus, with respect to $A^{\oplus}$, predicting the first continuation bit (corresponding to symbol 0) at length N is equivalent to predicting membership in L_A.

Lemma 10. *[Padded Construction] From a Boolean Acyclic DFA A we can construct, in time polynomial in $|A| + N$, a DFA $A^{\oplus}$ with $L_{A^{\oplus}} \subseteq \{0,1\}^{N+1}$ and*

$$\text{for } u \in \{0,1\}^N \text{ and } b \in \{0,1\}: \quad u \cdot b \in L_{A^{\oplus}} \iff \big(A(u) = 1\big) \text{ or } \big(b = 1\big). \quad (6.3)$$

Moreover, for every prefix y with $|y| < N$,

$$\big(\varphi(y,0),\ \varphi(y,1),\ L_{A^{\oplus}}(y)\big) = (1, 1, 0),$$

and for every $u \in \{0,1\}^N$,

$$\big(\varphi(u,0),\ \varphi(u,1),\ L_{A^{\oplus}}(u)\big) = \big(A(u),\ 1,\ 0\big).$$

The construction adds at most $N+1$ states.

Proof. Minimize the DFA A if it is not minimal. Create new states $q'_1, \ldots, q'_N$ and a new accepting state q_{fin}. For $1 \le i < N$ set

$$\delta_{A^{\oplus}}(q'_i, 0) = q'_{i+1}, \qquad \delta_{A^{\oplus}}(q'_i, 1) = q'_{i+1},$$

and at $i = N$ set

$$\delta_{A^{\oplus}}(q'_N, 1) = q_{\text{fin}}, \qquad \delta_{A^{\oplus}}(q'_N, 0) = q_{\text{dead}}.$$

From q_{fin} send both symbols to q_{dead} (so acceptance occurs only at length $N+1$). Now modify A to create $A^{\oplus}$ as follows. For each non-dead state q with $\ell(q) < N$ and each $\sigma \in \{0,1\}$:

- If $\delta_A(q, \sigma) = q_{\text{dead}}$ in A, set $\delta_{A^{\oplus}}(q, \sigma) = q'_{\ell(q)+1}$ (redirect the dead transition into the chain).
- Otherwise set $\delta_{A^{\oplus}}(q, \sigma) = \delta_A(q, \sigma)$.

For each state q at depth N set

$$\delta_{A^\oplus}(q,1) = q_{\text{fin}}, \qquad \delta_{A^\oplus}(q,0) = \begin{cases} q_{\text{fin}}, & q \in F_A, \\ q_{\text{dead}}, & q \notin F_A. \end{cases}$$

See Figure 6.2 for a simple example construction for a Boolean Acyclic DFA computing a conjunction.

All transitions out of q_{dead} point to q_{dead}, i.e., $\delta_{A^\oplus}(q_{\text{dead}}, \sigma) = q_{\text{dead}}$ for both symbols. (If A recognises the empty language, then $Q = \{q_{\text{dead}}\}$; in this case we additionally introduce a fresh start state $\tilde{q}_0$ of depth 0 with $\delta_{A^\oplus}(\tilde{q}_0, 0) = \delta_{A^\oplus}(\tilde{q}_0, 1) = q'_1$ and take $\tilde{q}_0$ as the start state; the conclusions below still hold.)

By construction, acceptance can occur only at q_{fin} after exactly $N{+}1$ symbols, so $L_{A^\oplus} \subseteq \{0,1\}^{N+1}$. The rule at depth N gives equation 6.3. For any prefix y with $|y| < N$, either an original transition still has an accepting continuation, or a dead transition is redirected to the chain $q'_{|y|+1} \to \cdots \to q'_N \to q_{\text{fin}}$ (taking the last symbol 1). Hence $\varphi(y,0) = \varphi(y,1) = 1$ and $L_{A^\oplus}(y) = 0$ for any y with $|y| < N$. At depth N, for every $u \in \{0,1\}^N$ our construction enforces

$$\delta_{A^\oplus}\big(\,\delta_{A^\oplus}(q_0,u),1\big) = q_{\text{fin}} \quad \text{and} \quad \delta_{A^\oplus}\big(\,\delta_{A^\oplus}(q_0,u),0\big) = \begin{cases} q_{\text{fin}}, & \delta_A(q_0,u) \in F_A, \\ q_{\text{dead}}, & \delta_A(q_0,u) \notin F_A, \end{cases}$$

so $\varphi(u,1) = 1$, $\varphi(u,0) = A(u)$, and $L_{A^\oplus}(u) = 0$, which is exactly

$$\big(\varphi(u,0),\ \varphi(u,1),\ L_{A^\oplus}(u)\big) = \big(A(u),\ 1,\ 0\big).$$

□

Reduction from standard learning to NSP learning. Let D be any distribution on $\{0,1\}^N$. Define the *padded* distribution $D^\oplus$ on $\{0,1\}^{N+1}$ by sampling $u \sim D$ and returning $x := u \cdot 1$. By equation 6.3, $x \in L_{A^\oplus}$ for every u, so $D^\oplus$ is supported on positive examples as required by the NSP setting. Moreover, given a labeled standard example (u,y) with $y = A(u)$, the full NSP label vector $f_{A^\oplus}(x)$ for $x = u \cdot 1$ is computable from (u,y):

$$\begin{aligned} &\text{for } 0 \le \ell < N: \quad \big(\varphi(x_{:\ell},0), \varphi(x_{:\ell},1), L(x_{:\ell})\big) = (1,1,0), \\ &\text{for } \ell = N: \quad \big(\varphi(x_{:N},0), \varphi(x_{:N},1), L(x_{:N})\big) = (y,1,0), \\ &\text{for } \ell = N{+}1: \quad \big(\varphi(x_{:N+1},0), \varphi(x_{:N+1},1), L(x_{:N+1})\big) = (0,0,1). \end{aligned}$$

Here $x_{:\ell}$ denotes the length-ℓ prefix of the padded string x.

Theorem 16. *Fix N and a polynomial $p(\cdot)$. If $\mathrm{ADFA}^N_{p(N)}$ is efficiently PAC-learnable in the* NSP *setting from positive examples, then $\mathrm{ADFA}^N_{p(N)}$ is efficiently PAC-learnable in the conventional classification (binary-label) setting.*

Proof. Let $\mathcal{A}_{\mathrm{NSP}}$ be an efficient NSP learner for $\mathrm{ADFA}^N_{p(N)}$. From i.i.d. labeled samples $(u^{(i)}, y^{(i)})$ with $y^{(i)} = A(u^{(i)})$, form positive NSP examples $\big(x^{(i)}, f_{A^\oplus}(x^{(i)})\big)$ where $x^{(i)} = u^{(i)} \cdot 1$ using the rule above, and feed them to $\mathcal{A}_{\mathrm{NSP}}$. Let $\hat{f}$ be the returned predictor so that, with probability at least $1 - \delta$,

$$\mathcal{L}_{\mathrm{NSP}}\big(\hat{f};\, f_{A^\oplus},\, D^\oplus\big) = \mathbb{E}_{u\sim D}\Big[\mathbb{I}\big[\hat{f}(u\cdot 1) \neq f_{A^\oplus}(u\cdot 1)\big]\Big] \;\leq\; \epsilon.$$

Define a standard classifier $h : \{0,1\}^N \to \{0,1\}$ by

$$h(u) \;:=\; \text{the bit predicted by } \hat{f} \text{ for } \varphi(x_{:N}, 0) \text{ on } x = u\cdot 1.$$

Whenever $\mathbb{I}\big[\hat{f}(x) = f_{A^\oplus}(x)\big] = 1$, Lemma 10 gives $h(u) = A(u)$. Therefore

$$\Pr_{u\sim D}\big[h(u) \neq A(u)\big] \;\leq\; \Pr_{u\sim D}\Big[\hat{f}(u\cdot 1) \neq f_{A^\oplus}(u\cdot 1)\Big] \;\leq\; \epsilon,$$

yielding an efficient PAC learner in the standard setting. The state complexity of $A^\oplus$ is at most $p(N) + N + 1$, preserving polynomial time complexity. □

Corollary 16.1 (Cryptographic hardness for NSP learning of acyclic DFAs)**.** *Under the assumptions of Theorem 15, there is no polynomial-time weak learner for* $\mathrm{ADFA}_{p(\cdot)}$ *in the* NSP *setting. Otherwise, Theorem 16 would yield a polynomial-time weak learner in the standard setting, contradicting Theorem 15.*

Discussion. The theorem shows that richer supervision via NSP does not circumvent the computational barrier for learning regular languages: under standard assumptions, there is no polynomial-time weak learner for $\mathrm{ADFA}_{p(\cdot)}$ and hence for DFAs even in the NSP setting. This remains true when the learner receives both positive and negative examples with NSP labels, showing that such additional supervision does not mitigate these barriers. The hardness is *improper*: it rules out efficient learning even when the hypothesis need not be a DFA, and thus applies to neural models trained to match NSP labels.

6.6 Hardness of Learning with Membership Queries

Definition. A conventional membership query oracle $\mathrm{MQ} : \Sigma^* \to \{0,1\}$ takes an input string and returns whether it belongs to the target language or not. In the NSP

setting, the membership query oracle $\mathrm{MQ}_{\mathrm{nsp}} : \Sigma^* \to \{0,1\}^{(|\Sigma|+1)(|x|+1)}$ returns all the NSP labels for an input string x. Since it contains the membership label of the input in its label, it has strictly more information than the conventional MQ oracle.

To understand how additional labels could provide more information, consider the following. If for any pair of prefixes or strings x, y, the $|\Sigma|$ continuation labels $\varphi(x), \varphi(y)$ differ, then it implies that they lead to two distinct states in the target DFA. This is because disagreement in the continuation label implies that there exists a suffix s such that $L(x \cdot s) \neq L(y \cdot s)$ and by Lemma 6, they must lead to two different states.

There are some classes of functions that can be efficiently identified by $\mathrm{MQ}_{\mathrm{nsp}}$ but not by conventional MQ. For instance, consider the class of singleton Boolean functions $\mathcal{F}_1$ over $\{0,1\}^N$ which contains 2^N functions that accept exactly 'one' Boolean input of length N. Such functions cannot be identified with a polynomial number of conventional membership queries (see Angluin [8] for a general characterization). The main idea is that no matter which input in $\{0,1\}^N$ a learner queries, an adversary can decide to always return 0 as the label until the learner queries $2^N - 1$ inputs.

Learning Singletons with $\mathrm{MQ}_{\mathrm{nsp}}$. The membership query oracle in the NSP setting is more powerful in the sense that such singleton Boolean functions can be identified easily in polynomial time. To see how, consider the following procedure: Let $x^\star \in \{0,1\}^N$ be the only string accepted by the target $f^\star : \{0,1\}^N \to \{0,1\}$. A learner first queries $\mathrm{MQ}_{\mathrm{nsp}}(x)$ any $x \in \{0,1\}^N$ and checks the $|\Sigma|$ continuation labels for the first index corresponding to the empty prefix λ. That indicates the first bit in the string accepted by the target function. Let $x_1^\star$ be the first bit of the target string. The learner then queries any string starting with $x_1^\star$ and obtains the second bit. Similarly, it can iteratively query $\mathrm{MQ}_{\mathrm{nsp}}$ and obtain the string $x^\star$ with at most N $\mathrm{MQ}_{\mathrm{nsp}}$ queries.

While the membership query oracle in the NSP setting $\mathrm{MQ}_{\mathrm{nsp}}$ is strictly more powerful than the one in the conventional classification setting, we show that there are DFAs in the class DFA_n that cannot be efficiently identified with membership queries only in the NSP setting. Similar to the case of hardness of learning with DFAs, we will construct functions where the NSP labels are uninformative and the problem becomes as hard as learning with the conventional membership query oracle.

Suffix Language family. Consider the following family of languages. Let $\Sigma = \{0,1\}$ (the argument applies to any Σ with $|\Sigma| \geq 2$). Let $S = \{0,1\}^{N/2}$ be the set of Boolean strings of length exactly $N/2$. The suffix language family $\mathcal{L}_S$ contains $2^{N/2}$ languages, where for each $s \in \{0,1\}^{N/2}$, the language $L_s \in \mathcal{L}_S$ only accepts strings which end with the suffix s.

Each of the language $L_s \in \mathcal{L}_S$ can be represented by a DFA of size at most $N/2+1$. For any suffix or string s, create a state corresponding to each prefix of $s = s_0 \cdot s_1 \cdots s_{N/2}$ including the empty prefix $s_0 = \lambda$ which serves as the start state. Define transitions $\delta(s_{:k}, s_{k+1}) = s_{:k+1}$. For every other transition, if the symbol is s_1, then they go to the state $s_{:1}$ or else they go to s_0. From the first state, the shortest string that leads to the accept state is of length $N/2$. For every other state q_i in $q_1, \ldots, q_{N/2}$, the shortest accepting suffix is of length $\frac{N}{2} - i$.

Proposition 17. *The class of Suffix languages $\mathcal{L}_S$ cannot be identified in polynomial time with membership queries* $\mathrm{MQ}_{\mathrm{nsp}}$ *in the* NSP *setting.*

Proof. A key characteristic of any suffix language L_s is that an accepting string exists for any prefix x. For any string x, the string $x \cdot s$ is in the language L_S. Thus, the continuation label for any prefix of any string will always be $(\varphi(x, 0), \varphi(x, 1)) = (1, 1)$. Hence, the continuation labels are not informative. The only informative signals are the membership labels.

In the NSP setting, the oracle $\mathrm{MQ}_{\mathrm{nsp}}$ will provide the membership labels of every prefix. Suppose a learner makes m $\mathrm{MQ}_{\mathrm{nsp}}$ queries where the maximum length of the queried input strings is k. Then each query eliminates at most $(k+1) - \frac{N}{2} \leq k$ suffixes out of $2^{N/2}$ possible suffixes if all the membership labels are 0. Thus, in the worst case, the learner can eliminate at most $O(mk)$ suffixes or functions from the class. If both the number of queries m and input length k are polynomial in N, then they cannot identify the target s. Hence, either the number of queries must be exponential or the learner must use exponential computational steps. □

Since the suffix language with suffixes of size $n/2$ can be represented with DFAs with at most n states, Prop. 17 immediately implies that the class DFA_n cannot be identified in polynomial time with membership queries alone in the NSP setting even with the richer set of labels.

Boolean functions and Acyclic DFAs. A similar argument applies to the class of Boolean functions as well. Consider the class of functions $\mathcal{F}_{2^N-1}$ over $\{0,1\}^N$ which accept all bit strings of length N except one. It is straightforward to see that every function in that class can also be represented by Boolean Acyclic DFAs with exactly $N + 2$ states.

For such a class, the continuation labels will be unhelpful for all but one prefix of length $N - 1$. The same line of argument as earlier shows that each membership query can eliminate at most 1 function from the class. An adversary can decide to return membership labels 1 for every input query and choose the function based on

the inputs queried by the learner, and hence in the worst case, the learner must make $2^{N-1} - 1$ queries before it can identify the target functions.
This implies that certain classes of Boolean functions, as well as the class ADFA_n cannot be identified with a polynomial number of MQ_{nsp} queries in the NSP setting.

6.7 Learning with a Language Model Teacher

Given the hardness of learning with random examples or membership queries alone, we now study the learnability of DFAs in a relatively more powerful model based on the information one can conveniently obtain via blackbox access to language models such as neural sequence models or PDFAs.

Problem and Assumptions. Let LM be a language model which induces a distribution over strings $\mathcal{D}_{\text{LM}}$ (correspondingly $\mathcal{D}^{\mathcal{T}}_{\text{LM}}$ for a sampling strategy $\mathcal{T}$). Assume that the support of the distribution $\mathcal{D}_{\text{LM}}$ is a regular language and let $A^\star$ be the DFA that recognises the support $L_{A^\star}$. Given blackbox access to such a language model, we would like to find $\hat{A}$ such that $\Pr_{x\sim\mathcal{D}_{\text{LM}}}\left[f_{\hat{A}}(x) \neq f_{A^\star}(x)\right] \leq \epsilon$ with high probability. In this setting, a learner has access to two types of queries: *(i) Membership queries* $\text{MQ}(x) \in \{0, 1\}$ which return $A^\star(x)$, and *(ii) Generative queries* $\text{Gen}_{\mathcal{D}_{\text{LM}}}(\cdot)$ which take an input string or prompt x and generate a string s along with NSP labels based on the distribution $\mathcal{D}_{\text{LM}}$ conditioned on the prompt x. Note that both these queries can be simulated with blackbox access to the language model (cf. Sec. 6.7.3).

Approach. Broadly, we first sample a set X with m NSP-labeled examples from the distribution $\mathcal{D}_{\text{LM}}$ using Generative queries $\text{Gen}_{\mathcal{D}_{\text{LM}}}(\lambda)$ where λ denotes the empty string. We will then use an extension of the $\text{L}^\star$ algorithm to make use of the membership queries and generative queries to obtain a hypothesis DFA $\hat{A}$ in polynomial time such that $|\hat{A}| \leq |A^\star|$ and $\hat{A}$ is *consistent* with the NSP labels of all m examples. A standard Occam-style generalization bound then yields a PAC-guarantee for the learning problem.

6.7.1 L-star Preliminaries

We will first discuss the preliminaries of the $\text{L}^\star$ framework based on modern and minimal treatments [85, 43].
Let $\hat{A}$ be a hypothesis DFA constructed by a learner and let $A^\star$ be the target DFA. We will refer to a string x such that $\hat{A}(x) \neq A^\star(x)$ as a *counterexample* that is provided by an Equivalence query oracle or by disagreement with an example in a training set.

Access and Test words. In this setup, we will have a pair of sets (Q, T) where $Q \subseteq \Sigma^*$ is a set of *access words*, and $T \subseteq \Sigma^*$ is a set of *test words*. Both Q and T are always nonempty. Intuitively, the set Q will play the role of states, where it will have a string corresponding to every state of our DFA. The test words T act as distinguishing strings for the access words Q. Both the sets Q, T contain the empty string λ. We will now define the notion of T-equivalence.

Definition 2 (T-Equivalence)**.** For a non-empty set $T \subseteq \Sigma^*$ and a language L, two strings u, v are T-equivalent: $u \equiv_T v$, if for every string $s \in T$, the string $u \cdot s \in L$ if and only if $v \cdot s \in L$.

Closed and Separable. A pair (Q, T) is *closed* if for every access word $q \in Q$ and every symbol $\sigma \in \Sigma$, there exists an access word $q' \in Q$ such that $q \cdot \sigma \equiv_T q'$. A pair (Q, T) is defined to be *separable* if every two distinct access words $q, q' \in Q$ are not T-equivalent: $q \not\equiv_T q'$.

Constructing a DFA. Given a closed and separable pair (Q, T) and access to membership queries MQ, one can construct a DFA as follows.

- Set the $q_0 = \lambda \in Q$.
- For every $q \in Q$, add them to the set of accept states F if $\text{MQ}(q) = 1$.
- For any access word q and symbol $\sigma \in \Sigma$, let q' be the word such that $q \cdot \sigma \equiv_T q'$. Then set the transition $\delta(q, \sigma) = q'$. Since (Q, T) is closed, such a state or access word q' must exist, and separability implies uniqueness of such a word.

To prove the correctness of our algorithm, we use the following technical lemmas from L* without proof.

Lemma 11 (L* Fact [7])**.** *Let (Q, T) be a closed and separable pair of sets which is consistent with some language L. Let $A^\star$ be the minimal automaton that accepts L. Then, $|Q| \leq |A^\star|$.*

The above Lemma follows directly from Lemma 6 and the fact that we have a distinguishing string in T for each pair of strings in Q.

Lemma 12 (L* Fact [7])**.** *Let (Q, T) be a pair that is separable but not closed. Then, using at most $|Q||T|(|\Sigma| + 1)$ membership queries, and in time polynomial in $(|Q|, |T|, |\Sigma|)$, we can identify $q' \notin Q$, such that $(Q \cup \{q'\}, T)$ is separable.*

The following lemma is the crux of the original $L^\star$ algorithm in terms of how it updates the pair (Q, T) based on disagreement between the hypothesis DFA and a labelled example.

Lemma 13 ($L^\star$ Fact [7])**.** *Let (Q, T) be a closed and separable pair, and $\hat{A}$ be the associated DFA. Let $x \in \Sigma^*$ be a string such that $\hat{A}(x) \neq A^\star(x)$. Then using at most $|x|$ membership queries and in time polynomial in $|x|$, we can identify $q' \notin Q$ and $t' \notin T$ such that $(Q \cup \{q'\}, T \cup \{t'\})$ is separable.*

See Worrell [172] for a tutorial of $L^\star$ in the framework described here and the proofs of the precise Lemmas above.

6.7.2 Learning with Membership and Generative Queries

In the NSP setting, counterexamples may arise from continuation-labels rather than membership disagreements, so Lemma 13 does not apply as is. This is where $L^\star_{\text{nsp}}$ departs from $L^\star$: we use a different method to process counterexamples $x \in L_{A^\star}$ using continuation-label disagreements. The next lemma formalises this update rule for $L^\star_{\text{nsp}}$.

Lemma 14. *Let (Q, T) be closed and separable, and let $\hat{A}$ be the minimal DFA induced by (Q, T). Suppose there exists a string x with $f_{\hat{A}}(x) \neq f_{A^\star}(x)$. Then one can find $q' \notin Q$ and $t' \notin T$ such that $(Q \cup \{q'\}, T \cup \{t'\})$ is separable, using membership queries and at most one generative query in polynomial time. If a generative query is used, let y denote its output; otherwise, set $y = \lambda$. The total number of membership queries is at most $|x| + |y|$, and the running time is polynomial in $|x| + |y| + |Q|$.*

Proof. By definition, we have a string such that $A^\star(x) = 1$ and $f_{\hat{A}}(x) \neq f_{A^\star}(x)$. Thus, there is a prefix $x_{:n}$ of x at which the NSP labels disagree; the disagreement is either (a) in the *membership* label for $x_{:n}$, or (b) in one of the *continuation* labels for some symbol $\sigma \in \Sigma$. We treat these two cases separately. For all cases, our goal will be to construct a string x' such that we can invoke Lemma 13 to obtain q' and t'.

Case A: Membership-label disagreement. There exists a prefix $x_{:n}$ of x such that $A^\star(x_{:n}) \neq \hat{A}(x_{:n})$. In this case we may simply take $x' = x_{:n}$ and apply Lemma 13. That lemma produces an access word $q' \notin Q$ and a test word $t' \notin T$ in time polynomial in $|x'|$ (and using at most $|x'|$ membership queries), and it guarantees that $(Q \cup \{q'\}, T \cup \{t'\})$ is separable.

Case B: Continuation-label disagreement. There exists a prefix $x_{:n}$ of x and a symbol $\sigma \in \Sigma$ such that

$$\varphi_{\hat{A}}(x_{:n}, \sigma) \neq \varphi_{A^\star}(x_{:n}, \sigma).$$

We distinguish two subcases according to the value of the target continuation label.
Subcase B1: $\varphi_{A^\star}(x_{:n}, \sigma) = 0$ *and* $\varphi_{\hat{A}}(x_{:n}, \sigma) = 1$. By the semantics of the continuation labels for the target, $\varphi_{A^\star}(x_{:n}, \sigma) = 0$ means that for every suffix $s \in \Sigma^*$ we have $A^\star(x_{:n} \cdot \sigma \cdot s) = 0$. On the other hand, $\varphi_{\hat{A}}(x_{:n}, \sigma) = 1$ means that the state $\tilde{q} := \delta_{\hat{A}}(q_0, x_{:n} \cdot \sigma)$ that $\hat{A}$ reaches after traversing $x_{:n} \cdot \sigma$ is not the dead state.
Therefore, we can search within $\hat{A}$ from $\tilde{q}$ to see whether there is some accepting continuation. Run a breadth–first search in $\hat{A}$ starting at $\tilde{q}$; if the search reaches a final state $q \in F_{\hat{A}}$, let s' be a shortest suffix labeling such a path. Then by construction, the length $|s'| \leq |Q|$ and

$$A^\star(x_{:n} \cdot \sigma \cdot s') = 0 \qquad \text{and} \qquad \hat{A}(x_{:n} \cdot \sigma \cdot s') = 1,$$

so $x' := x_{:n} \cdot \sigma \cdot s'$ is a standard membership counterexample. Applying Lemma 13 to x' yields $q' \notin Q$ and $t' \notin T$ in polynomial time.
Subcase B2: $\varphi_{A^\star}(x_{:n}, \sigma) = 1$ *and* $\varphi_{\hat{A}}(x_{:n}, \sigma) = 0$. The target label $\varphi_{A^\star}(x_{:n}, \sigma) = 1$ asserts that there exists a suffix $s' \in \Sigma^*$ with $A^\star(x_{:n} \cdot \sigma \cdot s') = 1$. In contrast, $\varphi_{\hat{A}}(x_{:n}, \sigma) = 0$ means that $\delta_{\hat{A}}(\delta_{\hat{A}}(q_0, x_{:n}), \sigma) = q_{\text{dead}}$, hence $\hat{A}(x_{:n} \cdot \sigma \cdot s) = 0$ for every suffix s.
As a consequence of Prop. 17, we also have that such a suffix cannot be found in polynomial time using membership queries only. Thus, to find an accepting suffix, we call the generative query oracle with input $x_{:n} \cdot \sigma$. Since $\varphi_{A^\star}(x_{:n}, \sigma) = 1$, such a suffix is guaranteed to exist. Let $s' = \text{Gen}_{\mathcal{D}_{\text{LM}}}(x_{:n} \cdot \sigma)$ and we have $x' = x_{:n} \cdot \sigma \cdot s'$ which is rejected by $\hat{A}$ (since it reaches a dead state after reading $x_{:n} \cdot \sigma$). Thus, invoking Lemma 13 on x' produces the desired q' and t' in polynomial time.
In all cases we obtain $q' \notin Q$ and $t' \notin T$ such that $(Q \cup \{q'\}, T \cup \{t'\})$ is separable. This completes the proof. □

Algorithm. Given a set of m NSP-labelled examples, the $\text{L}^\star_{\text{nsp}}$ algorithm works as follows. It starts with $Q = T = \{\lambda\}$ and iteratively updates the pair (Q, T) such that they are always separable. For a hypothesis $\hat{A}$ associated with (Q, T), it finds the first disagreement in the NSP labels with the examples in the training set X. Using membership queries and generative queries based on Lemma 14, it updates the pair (Q, T) and adds at least one state whenever there is a disagreement with the training examples. The closure step remains the same as in the original $\text{L}^\star$ algorithm. The

Algorithm 2 $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm

$Q \leftarrow \{\lambda\}, \quad T \leftarrow \{\lambda\}$
$X = \langle x^{(i)}, f_{A^\star}(x^{(i)})\rangle_{i=1}^{m}$ ▷ call $\mathrm{Gen}_{\mathcal{D}_{\mathrm{LM}}}(\lambda)$ m times
while (Q, T) not closed **do**
    Use Lemma 12 to obtain q'; $Q \leftarrow Q \cup \{q'\}$
end while
Construct hypothesis automaton $\hat{A}$ from (Q, T)
while $\hat{A}$ not consistent with X **do**
    Choose $x \in X$ with $f_{\hat{A}}(x) \neq f_{A^\star}(x)$
    while $f_{\hat{A}}(x) \neq f_{A^\star}(x)$ **do** ▷ Use Lemma 14
        Let $x_{:n}$ be the prefix with first NSP mismatch
        let σ denote the symbol if it is a continuation mismatch
        if membership mismatch at $x_{:n}$ **then** ▷ Case A
            $w' \leftarrow x_{:n}$
        else if continuation mismatch with $A^\star$ forbids and $\hat{A}$ admits **then** ▷ Case B1
            $s' \leftarrow$ find $\hat{A}$-accepting suffix from $x_{:n} \cdot \sigma$
            $w' \leftarrow x_{:n} \cdot \sigma \cdot s'$
        else ▷ Case B2
            $w' \leftarrow x_{:n} \cdot \sigma \cdot \mathrm{Gen}_{\mathcal{D}_{\mathrm{LM}}}(x_{:n} \cdot \sigma)$
        end if
        Use Lemma 13 on w' to obtain q' and t'
        $Q \leftarrow Q \cup \{q'\}, \; T \leftarrow T \cup \{t'\}$
        while (Q, T) not closed **do**
            Use Lemma 12 to obtain q'; $Q \leftarrow Q \cup \{q'\}$
        end while
        Reconstruct $\hat{A}$ from (Q, T)
    end while
end while

algorithm terminates when $\hat{A}$ induced by (Q, T) is consistent with all the training examples. Since $|Q|$ is guaranteed to be at most $|A^\star|$, the algorithm must terminate after at most $|A^\star|$ disagreements with the training examples. The pseudocode of the $\mathrm{L}^{\star}_{\mathrm{nsp}}$ is given in Algorithm 2.

Theorem 18. *Let $A^\star$ be the minimal DFA that recognises the support language $L_{A^\star} \subseteq \Sigma^*$ of the distribution $\mathcal{D}_{\mathrm{LM}}$. Given m* NSP *labelled examples $X = \langle x^{(i)}, f_{A^\star}(x^{(i)})\rangle_{i=1}^{m}$, with access to membership query oracle* MQ *and generative query oracle* $\mathrm{Gen}_{\mathcal{D}_{\mathrm{LM}}}$ *with respect to $\mathcal{D}_{\mathrm{LM}}$, Algorithm 2 outputs a DFA $\hat{A}$ such that $f_{\hat{A}}(x^{(i)}) = f_{A^\star}(x^{(i)})$ for all $i = 1, \ldots, m$. The running time is polynomial in $|A^\star|, |\Sigma|, \max_i |x^{(i)}|$, and the length of the longest string returned by the generative query oracle.*

Proof. If the target language is $L_{A^\star} = \emptyset$, then the initial hypothesis DFA $\hat{A}$ at Line 6 will also be such that $\hat{A}(x) = 0$ for all $x \in \Sigma^*$. Whenever a disagreement with any training example is found, by Lemma 14, the algorithm will add at least one string to Q and effectively identify at least one new state from $A^\star$. By Lemma 11, we have that $|Q| \leq |A^\star|$ and thus after at most $|A^\star|$ iterations of the main loop (Line 7) or equivalently, at most $|A^\star|$ calls to Lemma 13, we have that $|Q| = |A^\star|$. Each state in Q is uniquely identified with a set of distinguishing strings that are consistent with $A^\star$. Since (Q, T) is closed and the transitions are defined using membership queries to check T-equivalence, the final DFA is isomorphic to the target DFA $A^\star$. Hence, in such a case $\hat{A} = A^\star$ and the hypothesis must be consistent with all training examples. Thus, it will terminate. The algorithm can, of course, terminate with $|Q| < |A^\star|$ if the hypothesis DFA is consistent with all the training examples.

□

As a consequence of the above theorem and the fact that $\log |\mathrm{DFA}_n| = O(n \log n)$ (Lemma 8), Theorem 19 follows from standard Occam's razor arguments.

Theorem 19. *Let $A^\star \in \mathrm{DFA}_n$ be any minimal DFA with at most n states, and let $\mathcal{D}_{\mathrm{LM}}$ be a distribution over strings whose support is $L_{A^\star}$. There exists an algorithm with access to the membership query oracle* MQ *and generative query oracle* $\mathrm{Gen}_{\mathcal{D}_{\mathrm{LM}}}$ *producing* NSP *labeled examples in $L_{A^\star}$, that runs in time polynomial in $n, 1/\epsilon, 1/\delta$ and the length of the largest string produced by the generative query oracle, and with probability at least $1 - \delta$, outputs a DFA $\hat{A}$ such that,*

$$\mathcal{L}_{\mathrm{NSP}}(f_{\hat{A}}; f_{A^\star}, \mathcal{D}_{\mathrm{LM}}) = \Pr_{x \sim \mathcal{D}_{\mathrm{LM}}} \left[f_{\hat{A}}(x) \neq f_{A^\star}(x) \right] \leq \epsilon.$$

Discussion. A key benefit of this model of learning is that the guarantee we get is with respect to a desired distribution. For example, if one uses the L* algorithm to learn DFAs using positive and negative examples, the target distribution (on negative examples) is often unclear and can be artificial. In practice, for generative models, we often do not have access to the target distribution on *negative* examples. For the purpose of identifying the support of language models, arguably the most relevant distribution is the one induced by the model itself, e.g., if one intends to predict whether the language model will generate erroneous strings.

6.7.3 Simulating Membership and Generative Queries

We describe how membership and generative queries can be conveniently simulated by blackbox access to the target language model LM. For a neural language model, the empty string corresponds to [BOS] token.

(i) *Membership Queries.* Given a query string $x = w_1 \cdots w_N \in \Sigma^*$, compute the NSP continuation labels for every prefix of x using the language model including the empty string λ. For a neural language model, this is simply done by obtaining the next token probabilities for each prefix and applying the truncation rule $\mathcal{T}$ (cf. Sec. 6.2.2). If for any prefix the continuation label $\varphi(x_{:n}, w_{n+1}) = 0$ then w_{n+1} is not a valid continuation, so the oracle can return 0. If the path traversed by $w_1, \cdots, w_N$ is valid, then we can check whether $\varphi(x, \textsf{[EOS]})$ is 1 or 0 and thus $\mathrm{MQ}(x) = \varphi(x, \textsf{[EOS]})$.

(ii) *Generative Queries.* For any string or prefix x, the Generative Query Oracle $\mathrm{Gen}_{\mathcal{D}_{\mathrm{LM}}}(x)$ provides $(s, \boldsymbol{y})$ where s is a string from the distribution $\mathcal{D}^{\mathcal{T}}_{\mathrm{LM}}$ conditioned x. If no such continuation exists and x is at a dead state, then it returns 'None'. To simulate the example oracle or equivalently, to obtain examples from the target distribution $\mathcal{D}^{\mathcal{T}}_{\mathrm{LM}}$, one can simply query the oracle with empty string $\mathrm{Gen}_{\mathcal{D}_{\mathrm{LM}}}(\lambda)$. To obtain a continuation for a particular prefix x, one can provide it as an input prompt to the language model and first check whether x traverses a valid path based on the NSP labels as described above. If it does not, then return 'None'. If it does, then iteratively sample the next tokens using the truncation rule until the [EOS] is permissible $\varphi(x \cdot s, \textsf{[EOS]}) = 1$.

6.7.4 On Exact Learning with Membership and Equivalence Queries

We show that the class of DFAs can be learned exactly with membership queries and *two types* of equivalence queries. (i) $\mathrm{EQ}^{+}_{\mathrm{nsp}}(A; A^\star)$ which returns a positive NSP-labelled counterexample with at least one disagreement in NSP labels or else it returns equivalence; (ii) $\mathrm{EQ}_{\mathrm{pos}}(A; A^\star)$ which just returns a string with no additional labels, such that the string is accepted by $A^\star$ but rejected by A if there exists such a string, otherwise it returns none. The problem of learning with just membership and NSP equivalence queries remains open.

For the setting where a learner has access to two types of equivalence queries, the algorithm is almost the same as the one described earlier (Alg. 2) with a couple of differences. Similar to the previous algorithm with membership and generative queries, the learner will maintain a pair (Q, T) and iteratively update them, but with

counterexamples from $\mathrm{EQ}_{\mathrm{nsp}}$ instead of a set of labelled examples. There are only two changes to address, which we describe below.

(i) The first thing to note is that by Proposition 14, the equivalence query oracle $\mathrm{EQ}^{+}_{\mathrm{nsp}}$ is well-defined and always guaranteed to return a counterexample. If the target DFA $A^\star$ is such that $L_{A^\star} = \emptyset$, then the initial hypothesis DFA in Line 6 of Alg. 2 will be the same as $A^\star$ and thus, we do not need any counterexample.

(ii) Secondly, we do not have access to a generative query oracle anymore, so we cannot use it to find an accepting continuation when the target NSP label says that a suffix exists, but for our hypothesis DFA, no such suffix exists. The difficulty is that $\mathrm{EQ}^{+}_{\mathrm{nsp}}$ against our hypothesis need not return a counterexample beginning with the particular prefix $x_{:n} \cdot \sigma$. To force such a witness, we will use $\mathrm{EQ}_{\mathrm{pos}}(A, A^\star)$. Consider the DFA $A_{x_{:n}\sigma}$ whose language is

$$L_{A_{x_{:n}\sigma}} \;:=\; \Sigma^* \setminus \big(x_{:n} \cdot \sigma \cdot \Sigma^*\big),$$

i.e., $A_{x_{:n}\sigma}$ accepts exactly the strings that do *not* begin with the prefix $x_{:n}\cdot\sigma$. Submit the equivalence query $\mathrm{EQ}_{\mathrm{pos}}(A_{x_{:n}\sigma}; A^\star)$. Because $\varphi_{A^\star}(x_{:n}, \sigma) = 1$, there exists an accepted string with prefix $x_{:n} \cdot \sigma$; therefore, the oracle must return a positive counterexample of the form

$$x' \;=\; x_{:n} \cdot \sigma \cdot s' \quad \text{with} \quad A^\star(x') = 1.$$

But x' is rejected by $\hat{A}$ (the run reaches the dead state after reading $x_{:n} \cdot \sigma$), so x' is again a standard membership counterexample.

Correction Queries. In the above approach and Sec. 6.7, we use generative queries and the positive equivalence query oracle to find an accepting suffix. Becerra-Bonache et al. [16] proposed a learning model where the learner has access to something called correction queries (CQ). For a string $x \in \Sigma^*$, using the query $\mathrm{CQ}(x)$ will return the shortest accepting suffix (i) if x is not in the target language and (ii) if such an accepting suffix exists. If the x is in the target language, then the oracle can return an empty string, and if no accepting suffix exists, then CQ returns a special symbol not in the alphabet Σ. Thus, CQ provides the information provided by an MQ along with an accepting suffix when it exists. Becerra-Bonache et al. [16] analyze learnability with correction and equivalence query oracle in the conventional classification setting. CQ and generative queries are closely related in the sense that generative queries can be seen as a distribution-dependent version of CQ along with additional NSP labels. In the NSP setting, using the approaches discussed above, it is straightforward to see that the class of DFAs is exactly learnable from positive examples only with a correction query CQ and equivalence query oracle $\mathrm{EQ}^{+}_{\mathrm{nsp}}$.

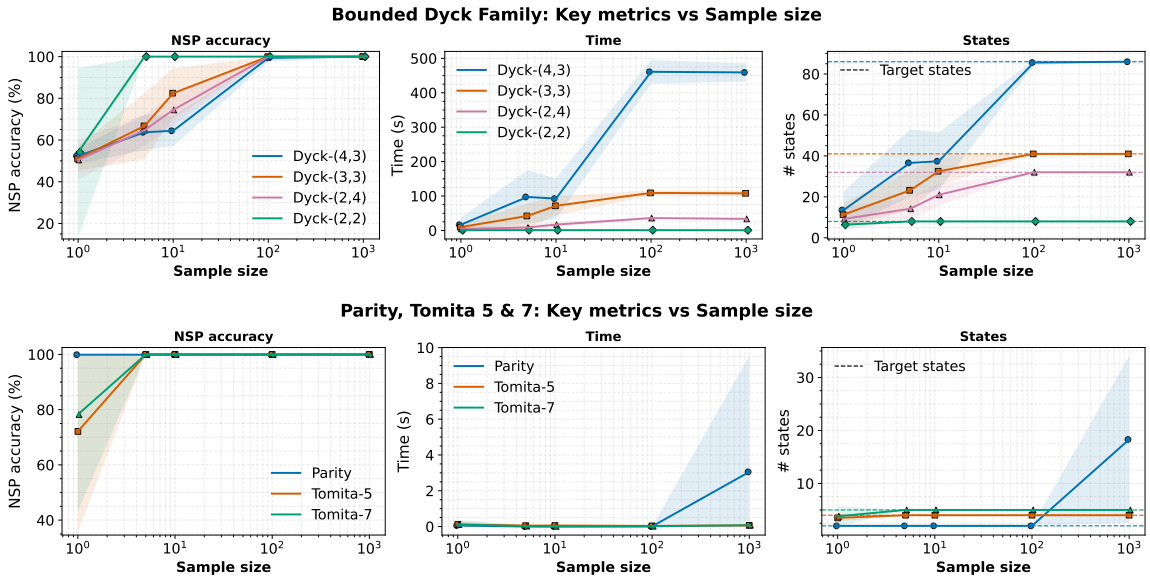


Figure 6.3: Key metrics for $\mathrm{L}^{\star}_{\mathrm{nsp}}$ across training-set sizes using a Transformer–LM teacher. Each point shows the mean over 10 trials; shaded regions denote standard deviation (see Sec. 6.8).

It is unclear whether exact learning with just membership and the NSP equivalence queries is feasible and is left as an open problem. When the learner gets a counterexample such that the only disagreement is of the form that $\varphi_{A^\star}(x_{:n}, \sigma) = 1$ and $\varphi_{\hat{A}}(x_{:n}, \sigma) = 0$, then one might wonder if a continuation can be recovered using the NSP membership query oracle $\mathrm{MQ}_{\mathrm{nsp}}$ which provides additional label. But as a consequence of Prop. 17, such an accepting continuation cannot be found in polynomial time using $\mathrm{MQ}_{\mathrm{nsp}}$ only.

6.8 Empirical Analysis

We evaluate the $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm as a tool for extracting DFAs from Transformer language models trained on regular languages. Given NSP-labeled strings from the model, we study how the NSP error $\mathcal{L}_{\mathrm{NSP}}$, the number of identified states, and the running time vary with the size of the training set. When the extracted automaton is not equivalent to the target DFA, we also find strings in the symmetric difference to identify *erroneous examples*. Further, we conduct ablations to analyze the effectiveness and usage of the continuation labels in the NSP setting.

Tasks. We consider 11 regular languages spanning 2–86 states: 6 Tomita grammars [155], Parity, and 4 bounded Dyck languages. Tomita grammars comprise small DFAs (2–5 states) and are commonly used as a benchmark for automata extraction [163, 167,

179]. Parity is the two-state language over $\Sigma = \{0, 1\}$ which contains all strings with an odd number of 1s. Bounded Dyck languages contain well-balanced parentheses up to a fixed depth and are regular. We denote by DYCK-(n, k) the Dyck language with n bracket types and depth at most k. The depth of a Dyck string is the maximum number of unclosed brackets in any prefix of the string. These languages have a relatively larger number of states: $\sum_{i=0}^{k} n^i + 1$. We use four bounded Dyck instances: DYCK-$(2, 2)$, DYCK-$(2, 4)$, DYCK-$(3, 3)$, and DYCK-$(4, 3)$. Further details of the tasks are in Sec. 6.9.

Setup. For each target language, we convert its canonical DFA to a PDFA to generate training strings. At non-final states, probability mass is split uniformly among transitions that avoid the dead state; at final states, generation terminates with probability t (otherwise, the next symbol is sampled as above). For each language, we choose t so that the empirical expected length is below 40.

Model training. We train Transformers as next-token predictors on sequences of the form $\texttt{[BOS]}\, s_1\, \texttt{[EOS]}\, s_2\, \texttt{[EOS]} \cdots$ with context window 250. Models use 8 layers and width 512, optimized with AdamW for up to 40k steps with early stopping. Our evaluation focuses on the support of the first string produced after [BOS]; the concatenated format matches standard training and provides additional learning signal.

DFA extraction. We use the trained Transformers to generate training sets of various sizes to evaluate the $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm. We create training sets with strings of length up to 80 that are generated and labelled using min-p sampling with threshold $p = 0.05$ as described in Sec. 6.2.2. The Transformer model serves as both membership and generative query oracle for $\mathrm{L}^{\star}_{\mathrm{nsp}}$. We evaluate sample sizes $\{1, 5, 10, 100, 1000\}$; for each size, we run 10 independent trials and report means and standard deviations. For small targets (e.g., Tomita), even a single positive example can be informative.

Results. Figure 6.3 summarises NSP accuracy, running time, and the number of extracted states for representative tasks. On Tomita, when teacher models are well trained, $\mathrm{L}^{\star}_{\mathrm{nsp}}$ recovers the target DFA quickly (often within a second; bottom row). For models that are not perfectly trained, such as for Parity, the algorithm extracts a much larger DFA and takes longer, depending on the number of target states. Note that since $\mathrm{L}^{\star}_{\mathrm{nsp}}$ adds a state only after finding a distinguishing string, the number of extracted states is always at most the number of states in the ground-truth DFA that recognises the language model's support. Thus, even when we recover more states than the target DFA, the result is faithful to the language teacher's support.

Bounded Dyck languages have a relatively much larger number of states (8–86 states). As shown in the top row of Figure 6.3, Dyck tasks naturally require more samples

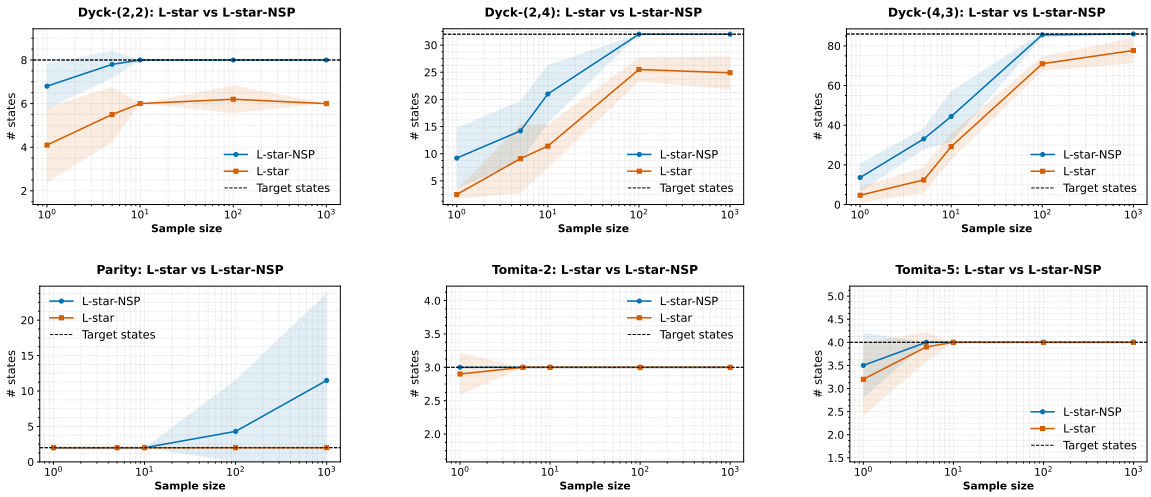


Figure 6.4: Comparison between binary labels ($L^\star$) and NSP labels ($L^\star_{\text{nsp}}$) by including negative examples by sampling from the untruncated distribution of the LM. Each point in the plots is the mean value over 10 trials; shaded regions show standard deviation. See Sec. 6.8 for details.

than Tomita, yet $L^\star_{\text{nsp}}$ converges to the target DFA and achieves near-perfect NSP accuracy within 100 examples. Our ablation experiments in Sec. 6.9.3 indicate that the continuation labels are heavily used and play a crucial role in identifying the states of the target (see Table 6.3).

Identifying erroneous examples. When the learned DFA $\hat{A}$ is not equivalent to the target DFA $A^\star$, we construct the product DFA B which recognises the strings in the symmetric difference of the two languages $L(B) = L(\hat{A}) \triangle L(A^\star)$. We use a BFS-like approach to identify several erroneous examples for the language model. Table 6.2 illustrates some erroneous examples for Bounded Dycks, Parity, and Tomita-5 language with LMs that weren't perfectly trained. Fig. 6.6 and 6.7 depict the extracted automaton for Parity and Tomita-5; the ones for DYCK-$(2,2)$ and DYCK-$(3,3)$ are too large to be visually informative. Note that these models were not intentionally trained to fail, and all the examples generated by the language models were in their respective target languages. The DFAs extracted by $L^\star_{\text{nsp}}$ were based on a few disagreements in the NSP labels of the generated strings. Training the language models for longer avoids such errors for synthetic languages of this scale. Note that the Transformer models used for Tomita-5 and Dyck languages in Figure 6.3 (well-trained) and Table 6.2 (imperfect) are different. See Sec. 6.9.1 for further details.

NSP Label vs Binary Label Ablations. To assess the value of NSP labels, we compare binary labels (classical $L^\star$) with NSP labels ($L^\star_{\text{nsp}}$ extension) on six languages. We sample strings from the model's *untruncated* distribution (actual next-token

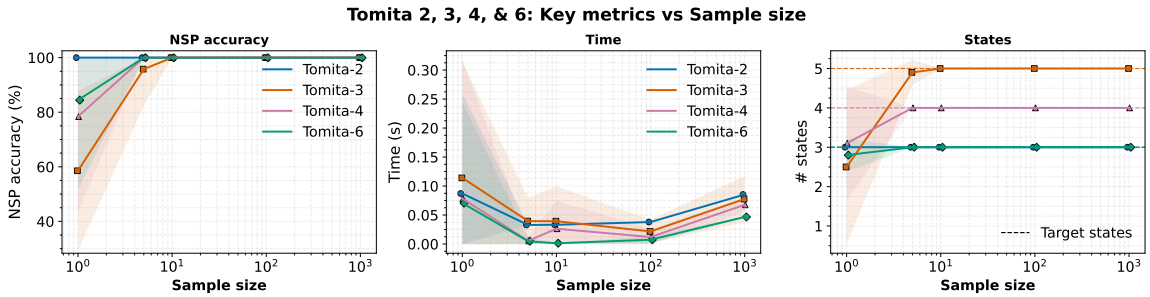


Figure 6.5: Key metrics for $L^\star_{nsp}$ on other Tomita grammars. See Sec. 6.8 for details.

probabilities) and label each as positive if it lies in the min-p truncated support $\mathcal{D}^{\mathcal{T}}_{\text{LM}}$ and negative otherwise. Naturally, most ($\approx 99.5\%$) samples are positive; nevertheless, even positive strings serve as counterexamples for $L^\star$ and using any natural variant of the distribution $\mathcal{D}_{\text{LM}}$ will have few negative examples.

Results. Figure 6.4 plots the number of extracted states versus sample size (means over 10 runs). On bounded Dyck languages, $L^\star_{nsp}$ reaches the target DFA with ≈ 10 examples for Dyck-$(2,2)$ and with ≈ 100 examples for Dyck-$(2,4)$ and Dyck-$(4,3)$, whereas $L^\star$ fails to recover the target even with 10^3 samples. For small Tomita DFAs, a single counterexample suffices, so both approaches perform similarly. For *Parity*, although the target DFA has 2 states, the teacher's support DFA is larger; with NSP labels, $L^\star_{nsp}$ identifies this larger support (enabling the discovery of erroneous strings), while binary labels yield no disagreements and thus no additional states. Because almost all samples are positive, classification accuracy is uninformative; predicting 1 leads to near perfect accuracy and thus the extracted state count is the meaningful signal. Note that the claim here isn't that $L^\star_{nsp}$ is superior (it is a direct extension of $L^\star$ itself). Rather, the goal here is to assess whether the additional labels are informative; the results indicate that leveraging NSP labels can be sample efficient for problems where the natural distribution primarily has positive examples.

6.9 Additional Ablations and Details of Experiments

We discuss additional details of the experimental setup as well as additional results for DFA extraction from language models.

Tasks. We provide additional details about the tasks considered in our experiments.

Tomita grammars. Table 6.1 provides the description of the 7 Tomita grammars [155] in the benchmark. We use all of them except the first one since it is trivial (1^*) and the results are uninformative. All the regular languages in the Tomita benchmark

Table 6.1: Tomita grammars and descriptions.

Tomita grammar	**Description**
Tomita 1	Strings of only 1s (including the empty string), 1^*.
Tomita 2	Repeated "10" pattern, $(10)^*$.
Tomita 3	Any block with an *odd* number of consecutive 1s is always followed by a block with an *even* number of consecutive 0s.
Tomita 4	Strings that do not contain `000` as a substring.
Tomita 5	Strings with an even number of 0s and an even number of 1s.
Tomita 6	Strings where the difference between the number of 0s and the number of 1s is a multiple of 3.
Tomita 7	$0^*1^*0^*1^*$ (zero or more 0s, then 1s, then 0s, then 1s).

have 2-5 states. These benchmarks have primarily been used to extract DFAs from RNN or Transformer-based classifiers [163, 167, 179].

Parity. The Parity language is a simple two-state DFA recognizing whether the number of 1s in a string is odd or even. This task has been widely studied in the context of Transformers, and previous works have shown empirical [18] and theoretical [70, 68] evidence to indicate that such functions are difficult for Transformers to model.

Bounded Dycks. DYCK-(n, k) represents the language with well-balanced parentheses with n types of brackets and depth at most k. These languages have also been widely used in analysis of sequence models [72, 19] since they capture hierarchical dependencies prevalent in natural languages. These DFAs have relatively larger number of states than Tomita grammars and Parity, and hence provide a testbed to evaluate learning algorithms on languages with increasing state complexity. We use the following languages in our experiments: DYCK-$(2, 2)$: 8 states, DYCK-$(2, 4)$: 32 states, DYCK-$(3, 3)$: 41 states, and DYCK-$(4, 3)$: 86 states.

Compute. The experiments in the work are not compute heavy. All our experiments were conducted using 8 NVIDIA Tesla V100 GPUs, each with 16GB of memory for training language models. Each run for up to 40k steps could take 1-6 hours depending on the task. Some runs are much shorter if the model achieves high accuracy quite early. The execution of $\mathrm{L}^\star_{\mathrm{nsp}}$ is primarily on CPU, and uses the GPU to answer Generative and Membership queries with the Transformer language model. The time taken for each run of $\mathrm{L}^\star_{\mathrm{nsp}}$ is provided in the main plots (Fig. 6.3 and 6.5). All of our implementations for language models are in PyTorch [121]. For our experiments with

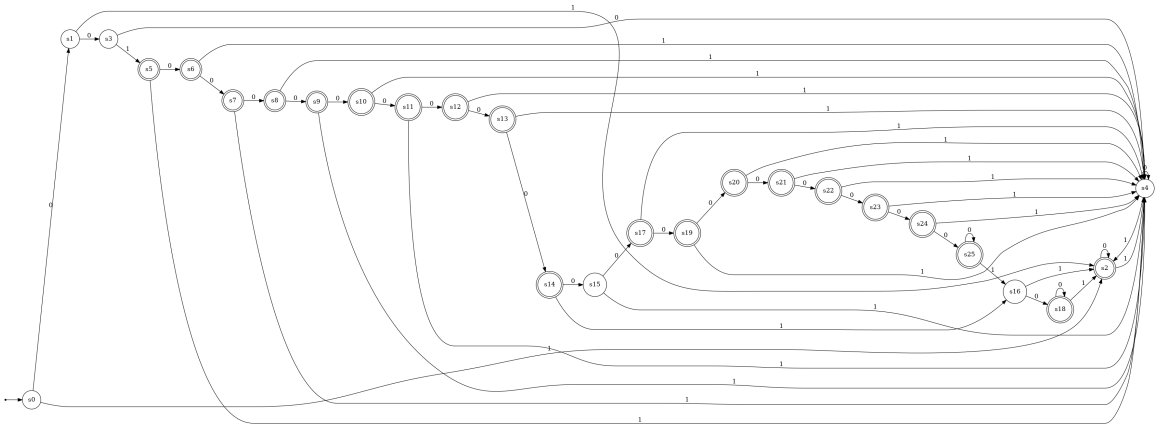


Figure 6.6: DFA with 26 states extracted by $\mathrm{L}^{\star}_{\mathrm{nsp}}$ from Transformer trained on Parity. See Sec. 6.9.1 for more details.

Transformers, we use the Huggingface Transformers library [171] with the GPT-2 [129] backbone.

6.9.1 Identifying Erroneous Examples

When the Transformer models trained on regular languages are not perfect, the extracted DFA is not isomorphic to the target DFA used to generate training data for Transformers.

There are two key things to note about this event. (i) When the extracted DFA has more states than the target DFA (such as for Parity in Fig. 6.3), then it implies that the support language for Transformer has at least as many states as the extracted DFA. This is because states are always created after identifying distinguishing strings (Lemma 6) using the teacher model. (ii) The cases where models are not perfectly trained, and our extracted DFA identifies erroneous strings, the models are still well-trained in the sense that all 1k strings generated using the language model are in the target language. Thus, the difference comes from disagreement in some NSP label, which the $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm then leverages to extract a bigger DFA, which is then used to identify erroneous strings.

Method. When the learned DFA $\hat{A}$ is not isomorphic to the target DFA $A^{\star}$, we construct the product automaton $B = \hat{A} \times A^{\star}$ where each state of B is a pair (p, q) with $p \in Q(\hat{A})$ and $q \in Q(A^{\star})$; on input symbol a, B transitions $\delta_B((p, q), a) = (\delta_{\hat{A}}(p, a), \delta_{A^{\star}}(q, a))$. The state (p, q) is accepting if and only if exactly one of p or q is accepting (XOR). The product DFA accepts the strings in the symmetric difference of the hypothesis and target DFA $L(B) = L(\hat{A}) \triangle L(A^{\star})$. We then enumerate erroneous

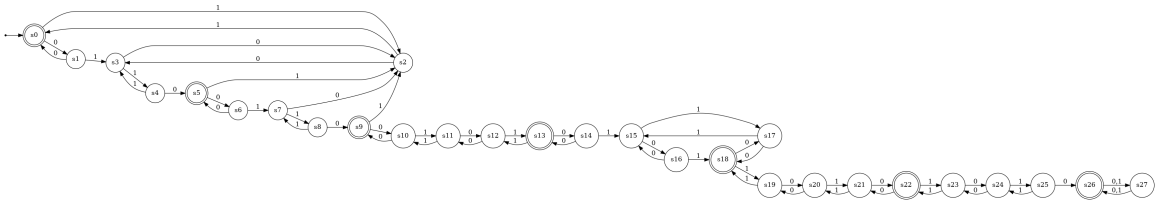


Figure 6.7: DFA with 28 states extracted by $\mathrm{L}^{\star}_{\mathrm{nsp}}$ from Transformer trained on Tomita-5. See Sec. 6.9.1 for more details.

strings by doing a BFS-like search within B to find accepting strings in nondecreasing length order.

Table 6.2: Examples of erroneous strings found via extracted DFAs. The $A^{\star}$ column within states denotes the number of states in the target DFA used to train the language model. The $\hat{A}$ column indicates the number of states in the extracted DFA. Ground truth labels are denoted by $y_{\star}$, and Teacher LM labels by y_T. Note: The models used as Teacher LM for Tomita-5, Dyck-$(2,2)$ and Dyck-$(3,3)$ for these results were imperfect and not the same models used for the experiments in Figure 6.3. See Sections 6.8 and 6.9.1 for more details.

Language	**States**		**Labels**		**Erroneous string** x
	$\lvert A^{\star}\rvert$	$\lvert\hat{A}\rvert$	$y_{\star}$	y_T	
Parity	2	26	1	0	0 0 1 0 0 0 0 0 0 0 0 0 0
			0	1	0 0 1 0 0 0 0 0 0 0 0 0 1 0
			0	1	0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
Dyck-(2,2)	8	189	0	1	( ) [ ] ( ) ( ) ( ) [ ] ( ) [ ]]
			0	1	[ ] [ ] ( ) ( ) ( ) [ ] ( ) [ ]]
			0	1	( ) [ ] ( ) ( ) ( ) [ ] ( ) [ ]] ( )
Dyck-(3,3)	41	87	0	1	{ { ( ) [ ] [ ] [ ] } ( ) { }
			0	1	{ { [ ] [ ] [ ] [ ] } ( ) { }
			0	1	{ { { } [ ] [ ] [ ] } ( ) { }
Tomita-5	4	28	0	1	0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1
			0	1	0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0
			0	1	0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 0 1

Results. We observed erroneous strings for languages like Parity, Tomita-5, Dyck-$(2,2)$, and Dyck-$(3,3)$. Examples of some erroneous strings identified by the hypothesis DFA are provided in Table 6.2. Figure 6.6 and 6.7 show the DFAs extracted for Parity and Tomita-5, respectively. The DFAs for Dyck-$(2,2)$ and Dyck-$(3,3)$ are too large to be visually interpretable. Constructing the product DFA is efficient, and identifying several erroneous examples takes only a few seconds. There is no natural distribution over the symmetric difference language, and further, it can even be finite in some cases which makes it difficult to systematically compute the accuracy of predicting

erroneous examples using the extracted DFA. The closest signal we have is the NSP accuracy for the extracted DFAs which is near perfect.

Fortunately, identifying strings in the symmetric difference language of $\hat{A}$ and $A^\star$ is quite efficient which can allow one to find numerous erroneous examples (if they exist). For instance, with 1k strings generated from a Transformer trained on Parity, the $\mathrm{L}^\star_{\mathrm{nsp}}$ algorithm extracted a DFA with 26 states within 10 seconds, and using the extracted DFA, we identified 1k strings in the symmetric difference language $L(\hat{A}) \triangle L(A^\star)$ in 16 seconds. Out of the 1000 strings identified by $\hat{A}$, 987 of them were erroneous: $A^\star(x) \neq \mathrm{MQ}(x)$ where $\mathrm{MQ}(x)$ denotes whether the string was in the support of the teacher language model and $A^\star(x)$ indicates whether the string was in the target regular language. Thus, such an approach can sometimes be far more efficient than finding erroneous strings by repeatedly sampling from a language model and testing whether they are correct or not.

For all languages except Parity, the language model became more robust when we retrained them for longer steps. The curves for Tomita-5 and the two Dyck languages in Fig. 6.3 are with retrained and more accurate Transformer models. We suspect that for the lengths considered in this work, retraining a larger Transformer on Parity for longer might make it more robust, but we keep the imperfect model to illustrate the differences in Fig. 6.3.

Table 6.3: Usage of different types of NSP labels by the $L^{\star}_{nsp}$ algorithm on various tasks. Refinement indicates the number of times states were added through disagreements (number of times the loop in Line 9 is executed or equivalently Lemma 14 is invoked). Mem fraction denotes the percentage of times prefix membership labels were used. B1 and B2 fractions denote how many continuation labels were used. See Sec. 6.9.3 for more details.

Language	States	Prefix Memberships (%)	Cont. B1 (%)	Cont. B2 (%)	Refinements
Sample size: 10					
Dyck22	8.0 ± 0.0	0.0 ± 0.0	16.7 ± 0.0	$\mathbf{83.3 \pm 0.0}$	6.00 ± 0.00
Dyck24	21.0 ± 5.4	0.0 ± 0.0	5.9 ± 2.6	$\mathbf{94.1 \pm 2.6}$	19.00 ± 5.35
Dyck33	32.5 ± 7.0	0.0 ± 0.0	3.5 ± 1.2	$\mathbf{96.5 \pm 1.2}$	30.50 ± 7.01
Dyck43	37.4 ± 14.1	0.0 ± 0.0	3.8 ± 3.2	$\mathbf{96.2 \pm 3.2}$	35.40 ± 14.14
Parity	2.0 ± 0.0	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	0.00 ± 0.00
Tomita 2	3.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	1.00 ± 0.00
Tomita 5	4.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	0.0 ± 0.0	0.0 ± 0.0	2.00 ± 0.00
Tomita 7	5.0 ± 0.0	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	0.0 ± 0.0	3.00 ± 0.00
Sample size: 100					
Dyck22	8.0 ± 0.0	0.0 ± 0.0	16.7 ± 0.0	$\mathbf{83.3 \pm 0.0}$	6.00 ± 0.00
Dyck24	32.0 ± 0.0	0.0 ± 0.0	3.3 ± 0.0	$\mathbf{96.7 \pm 0.0}$	30.00 ± 0.00
Dyck33	41.0 ± 0.0	0.0 ± 0.0	2.6 ± 0.0	$\mathbf{97.4 \pm 0.0}$	38.90 ± 0.32
Dyck43	85.6 ± 1.3	0.0 ± 0.0	1.2 ± 0.0	$\mathbf{98.8 \pm 0.0}$	83.60 ± 1.26
Parity	2.0 ± 0.0	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	0.00 ± 0.00
Tomita 2	3.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	1.00 ± 0.00
Tomita 5	4.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	0.0 ± 0.0	0.0 ± 0.0	2.00 ± 0.00
Tomita 7	5.0 ± 0.0	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	0.0 ± 0.0	3.00 ± 0.00
Sample size: 1000					
Dyck22	8.0 ± 0.0	0.0 ± 0.0	16.7 ± 0.0	$\mathbf{83.3 \pm 0.0}$	6.00 ± 0.00
Dyck24	32.0 ± 0.0	0.0 ± 0.0	3.3 ± 0.0	$\mathbf{96.7 \pm 0.0}$	30.00 ± 0.00
Dyck33	41.0 ± 0.0	0.0 ± 0.0	2.6 ± 0.0	$\mathbf{97.4 \pm 0.0}$	39.00 ± 0.00
Dyck43	86.0 ± 0.0	0.0 ± 0.0	1.2 ± 0.0	$\mathbf{98.8 \pm 0.0}$	84.00 ± 0.00
Parity	18.3 ± 15.7	$\mathbf{70.0 \pm 48.3}$	0.0 ± 0.0	0.0 ± 0.0	11.80 ± 11.72
Tomita 2	3.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	1.00 ± 0.00
Tomita 5	4.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	0.0 ± 0.0	0.0 ± 0.0	2.00 ± 0.00
Tomita 7	5.0 ± 0.0	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$	0.0 ± 0.0	3.00 ± 0.00

6.9.2 Practical Notes on the Algorithm

As mentioned in Sec. 6.10, if the DFA representing the support of the language model has a large number of states, then the extraction algorithm can be extremely slow. The key bottleneck for $L^{\star}_{nsp}$ is the same as $L^{\star}$, which is the closure step (Line 22 in Alg. 2). Even though it is polynomial, the time complexity $O(|Q||T||\Sigma|)$ blows up when both $|Q|$ and $|T|$ are in the order of thousands, since the process is sequential. We observe that when the models are partially or poorly trained, then the number of

states blows up, and the algorithm slows down significantly, failing to terminate even after a day.

The other step where the $L^{\star}_{nsp}$ algorithm might fail is if the assumptions about the learning problem are violated, in which case the use of generative queries to find accepting suffixes (Line 18 in Alg. 2) may not terminate. If the support language is not regular, then even though a continuation is permissible according to min-p/top-p sampling, the language model may still not terminate. However, we did not observe any instance of this problem in all our experiments. We suspect that since the language models are typically trained to predict [EOS] after a certain number of steps, they always terminate (the EOS token becomes permissible) after a certain number of inference steps.

6.9.3 Ablation: Usage of Continuation Labels

When we apply the $L^{\star}_{nsp}$ algorithm on positive examples, information about refinements or new states is provided by two types of labels: (i) prefix membership labels, and (ii) continuation labels. Within continuation labels, there are two cases as described in Lemma. 14. Whenever the algorithm finds one of the three types of disagreements in the NSP labels between the hypothesis DFA and the examples in its set, it refines the DFA and adds one more state to its hypothesis.

Results. We run the $L^{\star}_{nsp}$ algorithm on 8 representative languages and track the usage of the type of disagreement used to update its hypothesis. Table 6.3 depicts the usage of different types of disagreements on three different sample sizes averaged across 10 runs. The prefix membership column indicates the percentage of times the prefix membership labels were used. The Cont. B1 column represents the cases where the hypothesis predicted that a certain continuation is permitted, but the continuation label indicated that no such continuation is allowed in the target DFA. The Cont. B2 column represents the cases where the hypothesis predicts that continuation is forbidden, but the true labels indicate otherwise. This is the case where the generative query oracle is invoked to generate accepting suffixes. See Sec. 6.7 for the details of the algorithm.

The results indicate that continuation labels are heavily used for all Dyck languages. Such languages also contain several transitions with dead states, and the results indicate that the algorithm is able to leverage continuation labels to identify new states. On parity and Tomita-5, the algorithm relies only on the prefix membership labels. This is natural since the DFAs for those languages do not have any dead states, and hence the continuation labels are uninformative. From positive examples, the

algorithm can obtain negative information from the prefix membership labels for such languages. Both Tomita-2 and Tomita-7 have a transition to a dead state. Since these are small DFAs, the algorithm converges to the target DFA with only a few refinements.

6.10 Future Work and Limitations

A natural question that remains open is the efficient learnability of DFAs with membership (MQ) and equivalence queries (EQ) in the NSP setting. In Sec. 6.7.4, we describe how DFAs can be learned exactly with membership queries and two types of equivalence queries, and discuss barriers to obtaining a standard MQ+EQ algorithm. Prop. 14 shows that NSP labels are sufficient, and a teaching (or characteristic) set exists. However, the size of such a set obtained from Prop. 14 is likely to be quite loose and could be improved.

Limitations. Even though the $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm is polynomial-time, it inherits some of the limitations of the $\mathrm{L}^{\star}$ framework it builds on, making it difficult to scale the approach to practical language models. In particular, when the target DFA (language model support) has a large number of states, the algorithm is quite slow. We observe that when models are poorly trained, the underlying DFA typically has thousands of states. The $\mathrm{L}^{\star}_{\mathrm{nsp}}$ algorithm identifies about 1k states, and the closure step is slow due to the $|Q||T||\Sigma|$ time complexity (discussed in more detail in Sec. 6.9.2). Further, while the factor of $|\Sigma|$ may not play a significant role for synthetic languages, it has immediate consequences for language models trained on text that have a large vocabulary. An interesting future work would be to develop more efficient system-level improvements to speed up the algorithm to make it applicable to relatively more practical language models.

Chapter 7

Conclusion and Open Problems

This dissertation studied the expressivity and learnability of modern sequence models through a combination of theoretical analysis and empirical investigation. While each chapter contains its own set of open questions, we close by discussing several future directions and high-level problems suggested by the thesis.

Our results in Chapter 3 establish sharp separations between the capabilities of Transformers and recurrent models, and they also clarify concrete limitations of one-layer Transformers. A particularly interesting and challenging direction is to understand the limitations of multi-layer Transformers. As discussed in Chapter 3, the techniques used to derive lower bounds for one-layer Transformers do not directly extend to two layers. Sanford et al. [135] provide conditional lower bounds for multi-layer Transformers, and more recently, a notable work by Chen et al. [33] proved unconditional lower bounds for multi-layer Transformers with causal masking and established depth separations. However, unconditional lower bounds for *vanilla* multi-layer Transformers (without causal masking) have remained elusive. A related goal is to develop lower-bound techniques beyond communication complexity that can better capture the structure of attention.

More generally, we still lack a clean characterisation of what log-width two-layer (or constant-depth) Transformers can compute. One open problem is whether constant-sized Transformers can compute all functions in the class TC^0, or whether they are restricted to a strict subclass. It is known [101] that constant-sized Transformers can only express functions within TC^0, and that with pause tokens they can express all functions in TC^0 [95]. Another related direction is to understand the capabilities and limitations of *hybrid* architectures that combine attention with recurrence. Using straightforward adaptations of the techniques in Chapter 3, one can obtain

communication-complexity-based lower bounds for a two-layer network with a recurrent layer followed by a Transformer layer. The same approach does not apply when the Transformer layer precedes the recurrent layer, and several natural computational questions about such models remain open.

Beyond the intrinsic complexity-theoretic interest of these questions, a central motivation is to produce insights that can meaningfully inform architecture design. Ideally, analyses on formal languages and algorithmic tasks would also help identify benchmarks that are genuinely diagnostic of architectural choices. A key question is which computational tasks serve as faithful proxies for real-world performance. For instance, while Transformers struggle with Parity [68, 18], one could argue that Parity-like tasks may not be representative of performance on practical applications such as summarisation, question answering, or code generation. A notable example in this direction is the work of Arora et al. [10], which shows that the synthetic multi-query associative-recall task is closely related to real-world LLM performance. In Chapter 4, we highlighted systematic differences in the empirical performance of Transformers and recurrent models on nearest-neighbour tasks, and connected these differences to practical in-context learning behaviour in Transformer-based LLMs. An important direction is to identify additional tasks of this kind—tasks that are simple and precise enough to study carefully, yet predictive of behaviour at scale.

Such tasks could also serve as small-scale proxy evaluations for architectural components. At present, obtaining a meaningful signal about a new architecture often requires training a large model (e.g., on the order of billions of parameters) on roughly a trillion tokens, which makes systematic iteration expensive. A promising direction is to curate a selective collection of existing synthetic tasks, identify new tasks, and build a benchmark that compares Transformers with the many recently proposed attention-free alternatives. A well-designed pipeline of this kind could allow researchers to test ideas at a smaller scale before committing to large-scale training runs.

In Chapter 5, we studied the learnability of a single-head attention-based model from queries. In contrast, the learnability of the same model from random examples is much less well-understood. More broadly, learning neural networks from random examples remains challenging in several regimes and is an active area of study, as discussed in Sec. 2.3.3. Within the query-learning setting, an interesting direction is to design methods that combine theoretical structure with practical heuristics and develop algorithms that are both effective and scalable, even if they lack provable guarantees to some degree. While related empirical approaches have been explored for feedforward

networks, attention introduces additional structure that may be exploitable. Our analysis in Chapter 5 also adopts an idealised setting in which the learner can query arbitrary sequences of vectors and the model is a regressor. This suggests several concrete follow-up questions, including the learnability of multi-head attention, query models that take sequences of tokens rather than vectors, and the effect of additional architectural components such as layer normalisation.

For the learnability of DFAs considered in Chapter 6, a natural extension is to ask whether one can learn probabilistic or weighted variants of DFAs using next-token probabilities from language models. Under the strong assumption that the target distribution can itself be captured by a PDFA, our algorithm from Chapter 6 can be modified in a relatively straightforward way to learn PDFAs. However, this is stronger than assuming that the *support* is regular, and it is unlikely to hold for realistic neural language models.

Even within the setting we consider, while our algorithm (in Chapter 6) is efficient in the polynomial-time sense, it is not yet practically scalable. A potentially more scalable approach is to train neural-parameterised PDFAs or HMMs with gradient descent by sampling sequences together with NSP labels from a target language model. Although heuristic, such an approach may scale better and enable learning automata with a substantially larger number of states, after which one could apply search algorithms to identify erroneous examples.

This line of work also raises a fundamental question about the role of extraction itself. While extracting interpretable objects is actively pursued as a way to better understand LLMs, even if we could extract a faithful DFA or a similar abstraction (e.g., a circuit or a program) for a real language model, how would such an object be useful in an actionable sense? For modern LLMs, any faithful abstraction is likely to be enormous (e.g., millions of states, or an impractically large program), and hence not directly human-interpretable. Making tangible progress on this question seems essential for connecting formal abstractions to practical understanding and control.

To conclude, this dissertation explored multiple facets of expressivity and learnability for sequence models, focusing on Transformers as the dominant modern architecture. In Chapter 3, we analysed differences in the expressivity of Transformers and recurrent models, and showed that problems such as Lookup, Equality, and Nearest Neighbour are easier to express with Transformers. Along the way, we discussed how tools such as almost-orthogonal vectors can be used to construct small-sized Transformers, and how communication-complexity-based arguments can be used to prove limitations of

recurrent models and one-layer Transformers. In Chapter 4, we empirically showed that Transformers outperform attention-free architectures on the Nearest Neighbour task, and studied the relation of the task to Transformer-based LLMs. Chapter 4 also explored broader questions around in-context learning, including interpretability, learning from more informative examples, and whether LLMs can act as learning algorithms.

Chapters 5 and 6 shift focus to theoretical questions about learnability in more powerful models of learning that involve queries. In Chapter 5, we gave provable methods for learning single-head attention-based models in various regimes. In Chapter 6, we theoretically analysed the learnability of automata in a new setting relevant to language models, and, beyond the complexity-theoretic analysis, gave a provable method to PAC-learn the truncated support of a language model. The algorithms in these two chapters are complementary. The method in Chapter 5 recovers the weights of the target model and therefore learns the true target function, but it is restricted to shallow one-layer regression models and requires query access to arbitrary sequences of vectors. The automata-learning algorithm in Chapter 6 is more flexible: it applies to any architecture, is designed for language models rather than regressors, and operates on sequences of symbols rather than vectors, which is closer to the query access provided by practical APIs. The trade-off is that the learned object is an automaton rather than the target neural network itself.

Although Chapters 5 and 6 differ from Chapters 3 and 4 in their core questions, they also contribute to our broader understanding of sequence models. Beyond its algorithmic contributions, Chapter 5 clarifies which components of a Transformer can be identified from input-output mappings alone, and in Chapter 6, we use the DFA extraction algorithm to identify failure modes of Transformers trained on regular languages.

Broadly, the first half of the dissertation characterised differences between Transformers and recurrent or state-space models, through both theoretical separations in expressivity and systematic gaps in empirical performance on algorithmic tasks. The second half developed new provable methods for learning attention-based models from black-box access to their outputs, both directly and through formal abstractions such as DFAs. We hope that the techniques and perspectives developed here will be useful both for sharpening our understanding of Transformers and for analysing new sequence models that emerge in the future.

Bibliography

[1] D. Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[2] A. V. Aho and J. D. Ullman. *The theory of parsing, translation, and compiling*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1972.

[3] K. Ahuja and D. Lopez-Paz. A closer look at in-context learning under distribution shifts. *arXiv preprint arXiv:2305.16704*, 2023.

[4] E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou. What learning algorithm is in-context learning? investigations with linear models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0g0X4H8yN4I.

[5] D. Angluin. On the complexity of minimum inference of regular sets. *Information and control*, 39(3):337–350, 1978.

[6] D. Angluin. Inductive inference of formal languages from positive data. *Information and control*, 45(2):117–135, 1980.

[7] D. Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[8] D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[9] L. Arnaboldi, B. Loureiro, L. Stephan, F. Krzakala, and L. Zdeborova. Asymptotics of sgd in sequence-single index models and single-layer attention networks. *arXiv preprint arXiv:2506.02651*, 2025.

[10] S. Arora, S. Eyuboglu, A. Timalsina, I. Johnson, M. Poli, J. Zou, A. Rudra, and C. Re. Zoology: Measuring and improving recall in efficient language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LY3ukUANko.

[11] D. Ataee Tarzanagh, Y. Li, X. Zhang, and S. Oymak. Max-margin token selection in attention mechanism. *Advances in neural information processing systems*, 36:48314–48362, 2023.

[12] E. Avcu, C. Shibata, and J. Heinz. Subregular complexity and deep learning. *arXiv preprint arXiv:1705.05940*, 2017.

[13] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

[14] Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36, 2024.

[15] N. Barnfield, H. Cui, and Y. M. Lu. High-dimensional analysis of single-layer attention for sparse-token classification. *arXiv preprint arXiv:2509.25153*, 2025.

[16] L. Becerra-Bonache, A. H. Dediu, and C. Tîrnăucă. Learning dfa from correction and equivalence queries. In *International Colloquium on Grammatical Inference*, pages 281–292. Springer, 2006.

[17] R. C. Berwick, K. Okanoya, G. J. Beckers, and J. J. Bolhuis. Songs to syntax: the linguistics of birdsong. *Trends in cognitive sciences*, 15(3):113–121, 2011.

[18] S. Bhattamishra, K. Ahuja, and N. Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.576. URL https://aclanthology.org/2020.emnlp-main.576.

[19] S. Bhattamishra, K. Ahuja, and N. Goyal. On the practical ability of recurrent neural networks to recognize hierarchical languages. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages

1481–1494, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.129. URL https://aclanthology.org/2020.coling-main.129.

[20] S. Bhattamishra, A. Patel, and N. Goyal. On the computational power of transformers and its implications in sequence modeling. In R. Fernández and T. Linzen, editors, *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 455–475, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.conll-1.37. URL https://aclanthology.org/2020.conll-1.37.

[21] S. Bhattamishra, M. Hahn, P. Blunsom, and V. Kanade. Separations in the representational capabilities of transformers and recurrent architectures. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 36002–36045. Curran Associates, Inc., 2024. doi: 10.52202/079017-1135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/3f630b20b7b3ac76d3a0016fe29b6dc0-Paper-Conference.pdf.

[22] S. Bhattamishra, A. Patel, P. Blunsom, and V. Kanade. Understanding in-context learning in transformers and LLMs by learning to learn discrete functions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ekeyCgeRfC.

[23] S. Bhattamishra, M. Hahn, and V. Kanade. Automata learning and identification of the support of language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=L8SMNWsxfK.

[24] S. Bhattamishra, K. Shah, M. Hahn, and V. Kanade. Provably learning attention with queries. *arXiv preprint arXiv:2601.16873*, 2026.

[25] P. Blanchard, D. J. Higham, and N. J. Higham. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4): 2311–2330, 2021.

[26] D. Blasi, R. Cotterell, L. Wolf-Sonkin, S. Stoll, B. Bickel, and M. Baroni. On the distribution of deep clausal embeddings: A large cross-linguistic study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3938–3943, 2019.

[27] A. Blum and R. Rivest. Training a 3-node neural network is np-complete. *Advances in neural information processing systems*, 1, 1988.

[28] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[29] N. H. Bshouty. Exact learning from membership queries: Some techniques, results and new directions. In *International Conference on Algorithmic Learning Theory*, pages 33–52. Springer, 2013.

[30] T. T. Cai and A. Zhang. Rop: Matrix recovery via rank-one projections. *The Annals of Statistics*, 43(1):102–138, 2015. ISSN 00905364. URL http://www.jstor.org/stable/43556510.

[31] N. Carlini, D. Paleka, K. D. Dvijotham, T. Steinke, J. Hayase, A. F. Cooper, K. Lee, M. Jagielski, M. Nasr, A. Conmy, E. Wallace, D. Rolnick, and F. Tramèr. Stealing part of a production language model. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5680–5705. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/carlini24a.html.

[32] S. Chan, A. Santoro, A. Lampinen, J. Wang, A. Singh, P. Richemond, J. McClelland, and F. Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

[33] L. Chen, B. Peng, and H. Wu. Theoretical limitations of multi-layer transformer. *ArXiv*, abs/2412.02975, 2024. URL https://api.semanticscholar.org/CorpusID:274464787.

[34] S. Chen and Y. Li. Provably learning a multi-head attention layer. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1744–1754, 2025.

[35] S. Chen, A. Klivans, and R. Meka. Efficiently learning one hidden layer relu networks from queries. *Advances in Neural Information Processing Systems*, 34: 24087–24098, 2021.

[36] S. Chen, Z. Dou, S. Goel, A. Klivans, and R. Meka. Learning narrow one-hidden-layer relu networks. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 5580–5614. PMLR, 2023.

[37] D. Chiang and P. Cholak. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7654–7664, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.527. URL https://aclanthology.org/2022.acl-long.527.

[38] D. Chiang, P. Cholak, and A. Pillay. Tighter bounds on the expressivity of transformer encoders. In *International Conference on Machine Learning*, pages 5544–5562. PMLR, 2023.

[39] N. Chomsky. Syntactic structures. In *Syntactic Structures*. De Gruyter Mouton, 2009.

[40] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 35, pages 118–161. Elsevier, 1963.

[41] A. Clark and F. Thollard. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5(May):473–497, 2004.

[42] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland. Finite state automata and simple recurrent networks. *Neural Comput.*, 1(3):372–381, Sept. 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.3.372. URL https://doi.org/10.1162/neco.1989.1.3.372.

[43] T. Colcombet, D. Petrişan, and R. Stabile. Learning Automata and Transducers: A Categorical Approach. In C. Baier and J. Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183

of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-175-7. doi: 10.4230/LIPIcs.CSL.2021.15. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2021.15.

[44] D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei. Why can GPT learn in-context? language models secretly perform gradient descent as meta-optimizers. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.247. URL https://aclanthology.org/2023.findings-acl.247/.

[45] A. Daniely and E. Granot. An exact poly-time membership-queries algorithm for extracting a three-layer reLU network. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-CoNloheTs.

[46] S. De, S. L. Smith, A. Fernando, A. Botev, G. Cristian-Muraru, A. Gu, R. Haroun, L. Berrada, Y. Chen, S. Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[47] C. De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[48] G. Deletang, A. Ruoss, J. Grau-Moya, T. Genewein, L. K. Wenliang, E. Catt, C. Cundy, M. Hutter, S. Legg, J. Veness, and P. A. Ortega. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WbxHAzkeQcn.

[49] I. Diakonikolas, D. M. Kane, V. Kontonis, and N. Zarifis. Algorithms and sq lower bounds for pac learning one-hidden-layer relu networks. In *Conference on Learning Theory*, pages 1514–1539. PMLR, 2020.

[50] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui. A survey for in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[51] J. Ebrahimi, D. Gelda, and W. Zhang. How can self-attention networks recognize Dyck-n languages? In T. Cohn, Y. He, and Y. Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.384. URL https://aclanthology.org/2020.findings-emnlp.384.

[52] N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[53] R. Eyraud and S. Ayache. Distillation of weighted automata from recurrent neural networks using a spectral approach. *Machine Learning*, 113(5):3233–3266, 2024.

[54] C. Fefferman and S. Markel. Recovering a feed-forward net from its output. *Advances in neural information processing systems*, 6, 1993.

[55] M. Finlayson, K. Richardson, A. Sabharwal, and P. Clark. What makes instruction learning hard? an investigation and a new challenge in a synthetic environment. *arXiv preprint arXiv:2204.09148*, 2022.

[56] P. Fischer and H.-U. Simon. On learning ring-sum-expansions. *SIAM Journal on Computing*, 21(1):181–192, 1992.

[57] S. Garg, D. Tsipras, P. S. Liang, and G. Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

[58] R. Ge, J. D. Lee, and T. Ma. Learning one-hidden-layer neural networks with landscape design. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkwHObbRZ.

[59] F. A. Gers and E. Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[60] C. L. Giles, C. B. Miller, D. Chen, H.-H. Chen, G.-Z. Sun, and Y.-C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[61] S. Goel, A. Klivans, P. Manurangsi, and D. Reichman. Tight Hardness Results for Training Depth-2 ReLU Networks. In J. R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-177-1. doi: 10.4230/LIPIcs.ITCS.2021.22. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2021.22.

[62] E. M. Gold. Language identification in the limit. *Information and control*, 10 (5):447–474, 1967.

[63] E. M. Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[64] S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

[65] M. W. Goudreau, C. L. Giles, S. T. Chakradhar, and D. Chen. First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513, 1994.

[66] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

[67] A. Gupta, A. Gu, and J. Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

[68] M. Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. doi: 10.1162/tacl_a_00306. URL https://aclanthology.org/2020.tacl-1.11.

[69] M. Hahn and N. Goyal. A theory of emergent in-context learning as implicit structure induction. *arXiv preprint arXiv:2303.07971*, 2023.

[70] M. Hahn and M. Rofin. Why are sensitive functions hard for transformers? In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14973–15008, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.800. URL https://aclanthology.org/2024.acl-long.800/.

[71] Y. Hao, D. Angluin, and R. Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022. doi: 10.1162/tacl_a_00490. URL https://aclanthology.org/2022.tacl-1.46.

[72] J. Hewitt, M. Hahn, S. Ganguli, P. Liang, and C. D. Manning. RNNs can generate bounded hierarchical languages with optimal memory. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.156. URL https://aclanthology.org/2020.emnlp-main.156.

[73] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[74] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

[75] Y. Huang, Y. Cheng, and Y. Liang. In-context convergence of transformers. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[76] G. Jäger and J. Rogers. Formal language theory: refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1598): 1956–1970, 2012.

[77] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security 20)*, pages 1345–1362, 2020.

[78] T. S. Jayram, R. Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

[79] S. Jelassi, D. Brandfonbrener, S. M. Kakade, and E. Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

[80] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984. URL https://api.semanticscholar.org/CorpusID:117819162.

[81] F. Karlsson. Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 43(2):365–392, 2007.

[82] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[83] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 45(6):983–1006, 1998.

[84] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[85] M. J. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[86] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[87] J. F. Kolen and S. C. Kremer. *A field guide to dynamical recurrent networks*. John Wiley & Sons, 2001.

[88] S. A. Korsky and R. C. Berwick. On the computational power of rnns. *arXiv preprint arXiv:1906.06349*, 2019.

[89] L. Kruger, B. Garhewal, and F. Vaandrager. Lower bounds for active automata learning. In F. Coste, F. Ouardi, and G. Rabusseau, editors, *Proceedings of 16th edition of the International Conference on Grammatical Inference*, volume 217 of *Proceedings of Machine Learning Research*, pages 157–180. PMLR, 10–13 Jul 2023. URL https://proceedings.mlr.press/v217/kruger23a.html.

[90] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1996.

[91] S. Li, I. Zadik, and M. Zampetakis. On the hardness of learning one hidden layer neural networks. In G. Kamath and P.-L. Loh, editors, *Proceedings of The 36th International Conference on Algorithmic Learning Theory*, volume 272 of *Proceedings of Machine Learning Research*, pages 700–701. PMLR, 24–27 Feb 2025. URL https://proceedings.mlr.press/v272/li25a.html.

[92] Y. Li, M. E. Ildiz, D. Papailiopoulos, and S. Oymak. Transformers as algorithms: generalization and stability in in-context learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[93] B. Liu, J. Ash, S. Goel, A. Krishnamurthy, and C. Zhang. Exposing attention glitches with flip-flop language modeling. *Advances in Neural Information Processing Systems*, 36, 2024.

[94] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[95] C. London and V. Kanade. Pause tokens strictly increase the expressivity of constant-depth transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=eG5oh8l1WZ.

[96] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.556. URL https://aclanthology.org/2022.acl-long.556.

[97] R. Magen, S. Shang, Z. Xu, S. Frei, W. Hu, and G. Vardi. Benign overfitting in single-head attention. *arXiv preprint arXiv:2410.07746*, 2024.

[98] P. Manolios and R. Fanelli. First-order recurrent neural networks and deterministic finite state automata. *Neural Comput.*, 6(6):1155–1173, Nov. 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.6.1155. URL https://doi.org/10.1162/neco.1994.6.6.1155.

[99] P. Marion, R. Berthier, G. Biau, and C. Boyer. Attention layers provably solve single-location regression. In *The Thirteenth International Conference on*

Learning Representations, 2025. URL https://openreview.net/forum?id=DVlPp7Jd7P.

[100] F. Mayr, S. Yovine, M. Carrasco, F. Pan, and F. Vilensky. A congruence-based approach to active automata learning from neural language models. In F. Coste, F. Ouardi, and G. Rabusseau, editors, *Proceedings of 16th edition of the International Conference on Grammatical Inference*, volume 217 of *Proceedings of Machine Learning Research*, pages 250–264. PMLR, 10–13 Jul 2023. URL https://proceedings.mlr.press/v217/mayr23a.html.

[101] W. Merrill and A. Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

[102] W. Merrill, G. Weiss, Y. Goldberg, R. Schwartz, N. A. Smith, and E. Yahav. A formal hierarchy of RNN architectures. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.43. URL https://aclanthology.org/2020.acl-main.43/.

[103] W. Merrill, A. Sabharwal, and N. A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022. doi: 10.1162/tacl_a_00493. URL https://aclanthology.org/2022.tacl-1.49.

[104] G. A. Miller and N. Chomsky. Finitary models of language users. In R. D. Luce, R. R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology*, pages 419–492. John Wiley, 1963.

[105] S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 1–9, 2019.

[106] S. Min, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Noisy channel language model prompting for few-shot text classification. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5316–5330, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.365. URL https://aclanthology.org/2022.acl-long.365.

[107] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi. MetaICL: Learning to learn in context. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.201. URL https://aclanthology.org/2022.naacl-main.201/.

[108] N. N. Minh, A. Baker, C. Neo, A. G. Roush, A. Kirsch, and R. Shwartz-Ziv. Turning up the heat: Min-p sampling for creative and coherent LLM outputs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FBkpCyujtS.

[109] K. Murota, Y. Kanno, M. Kojima, and S. Kojima. A numerical algorithm for block-diagonal decomposition of matrix*-algebras. *Part I: proposed approach and application to semidefinite programming, to appear in Japan Journal of Industrial and Applied Mathematics*, 2007.

[110] E. Muškardin, B. K. Aichernig, I. Pill, and M. Tappler. Learning finite state models from recurrent neural networks. In *International Conference on Integrated Formal Methods*, pages 229–248. Springer, 2022.

[111] R. O'Donnell. Analysis of boolean functions, 2021. URL https://arxiv.org/abs/2105.10386.

[112] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[113] C. W. Omlin and C. L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural networks*, 9(1):41–52, 1996.

[114] OpenAI. Gpt-4 technical report, 2023.

[115] T. Orekondy, B. Schiele, and M. Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4954–4963, 2019.

[116] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR, 2023.

[117] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback, 2022.

[118] N. Palmer and P. W. Goldberg. Pac-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theoretical Computer Science*, 387(1):18–31, 2007.

[119] M. Panwar, K. Ahuja, and N. Goyal. In-context learning through the bayesian prism. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HX5ujdsSon.

[120] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[121] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[122] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[123] B. Peng, S. Narayanan, and C. Papadimitriou. On limitations of the transformer architecture, 2024.

[124] J. Pérez, J. Marinković, and P. Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGBdo0qFm.

[125] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM (JACM)*, 35(4):965–984, 1988.

[126] L. Pitt and M. K. Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. *Journal of the ACM (JACM)*, 40(1): 95–142, 1993.

[127] M. Poli, S. Massaroli, E. Nguyen, D. Y. Fu, T. Dao, S. Baccus, Y. Bengio, S. Ermon, and C. Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

[128] J. B. Pollack. The induction of dynamical recognizers. *Machine learning*, 7(2): 227–252, 1991.

[129] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[130] A. Rao and A. Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.

[131] J.-F. Raymond, P. Tesson, and D. Thérien. An algebraic approach to communication complexity. In *Automata, Languages and Programming: 25th International Colloquium, ICALP'98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*, pages 29–40. Springer, 1998.

[132] Y. Razeghi, R. L. Logan IV, M. Gardner, and S. Singh. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

[133] P. Rodriguez. Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural computation*, 13(9):2093–2118, 2001.

[134] C. Sanford, D. Hsu, and M. Telgarsky. Representational strengths and limitations of transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=36DxONZ9bA.

[135] C. Sanford, D. Hsu, and M. Telgarsky. Transformers, parallel computation, and logarithmic depth. *arXiv preprint arXiv:2402.09268*, 2024.

[136] L. Sennhauser and R. Berwick. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5414. URL https://www.aclweb.org/anthology/W18-5414.

[137] O. Shamir. Distribution-specific hardness of learning neural networks. *Journal of Machine Learning Research*, 19(32):1–29, 2018.

[138] H. Sheen, S. Chen, T. Wang, and H. H. Zhou. Implicit regularization of gradient flow on one-layer softmax attention. *arXiv preprint arXiv:2403.08699*, 2024.

[139] H. Shi, S. Gao, Y. Tian, X. Chen, and J. Zhao. Learning bounded context-free-grammar via lstm and the transformer: Difference and the explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8267–8276, 2022.

[140] Y. Shi, Y. Sagduyu, and A. Grushin. How to steal a machine learning classifier with deep learning. In *2017 IEEE International symposium on technologies for homeland security (HST)*, pages 1–5. IEEE, 2017.

[141] S. M. Shieber. Evidence against the context-freeness of natural language. In *Philosophy, language, and artificial intelligence*, pages 79–89. Springer, 1985.

[142] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449. ACM, 1992.

[143] H. T. Siegelmann, B. G. Horne, and C. L. Giles. Computational capabilities of recurrent narx neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215, 1997.

[144] D. Sivakumar. Algorithmic derandomization via complexity theory. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 619–626, 2002.

[145] N. Skachkova, T. Trost, and D. Klakow. Closing brackets with recurrent neural networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 232–239, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5425. URL https://www.aclweb.org/anthology/W18-5425.

[146] L. Song, S. Vempala, J. Wilmes, and B. Xie. On the complexity of learning neural networks. *Advances in neural information processing systems*, 30, 2017.

[147] H. Straubing. *Finite automata, formal logic, and circuit complexity*. Springer Science & Business Media, 2012.

[148] L. Strobl, W. Merrill, G. Weiss, D. Chiang, and D. Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

[149] Y. Sun, L. Dong, S. Huang, S. Ma, Y. Xia, J. Xue, J. Wang, and F. Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

[150] M. Suzgun, Y. Belinkov, S. Shieber, and S. Gehrmann. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence, Aug. 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-3905. URL https://www.aclweb.org/anthology/W19-3905.

[151] M. Suzgun, Y. Belinkov, and S. M. Shieber. On evaluating the generalization of LSTM models in formal languages. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 277–286, 2019. doi: 10.7275/s02b-4d91. URL https://www.aclweb.org/anthology/W19-0128.

[152] M. Suzgun, S. Gehrmann, Y. Belinkov, and S. M. Shieber. Memory-augmented recurrent neural networks can learn generalized dyck languages. *arXiv preprint arXiv:1911.03329*, 2019.

[153] D. A. Tarzanagh, Y. Li, C. Thrampoulidis, and S. Oymak. Transformers as support vector machines. *arXiv preprint arXiv:2308.16898*, 2023.

[154] Y. Tian, Y. Wang, B. Chen, and S. S. Du. Scan and snap: Understanding training dynamics and token composition in 1-layer transformer. *Advances in neural information processing systems*, 36:71911–71947, 2023.

[155] M. Tomita. Learning of construction of finite automata from examples using hill-climbing. rr: Regular set recognizer. Technical report, 1982.

[156] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[157] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao,

V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[158] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[159] F. Vaandrager, B. Garhewal, J. Rot, and T. Wißmann. A new approach for active automata learning based on apartness. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 223–243. Springer, 2022.

[160] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[161] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[162] J. Von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.

[163] Q. Wang, K. Zhang, A. G. O. II, X. Xing, X. Liu, and C. L. Giles. A comparative study of rule extraction for recurrent neural networks. *CoRR*, abs/1801.05420v2, 2018. URL http://arxiv.org/abs/1801.05420v2.

[164] Z. Wang, S. Wei, D. Hsu, and J. D. Lee. Transformers provably learn sparse token selection while fully-connected nets cannot. *arXiv preprint arXiv:2406.06893*, 2024.

[165] J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, and T. Ma. Larger language models do in-context learning differently, 2023.

[166] Z. Wei, X. Zhang, Y. Zhang, and M. Sun. Weighted automata extraction and explanation of recurrent neural networks for natural language tasks. *Journal of Logical and Algebraic Methods in Programming*, 136:100907, 2024.

[167] G. Weiss, Y. Goldberg, and E. Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning*, pages 5247–5256. PMLR, 2018.

[168] G. Weiss, Y. Goldberg, and E. Yahav. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2117. URL https://www.aclweb.org/anthology/P18-2117.

[169] G. Weiss, Y. Goldberg, and E. Yahav. Learning deterministic weighted automata with queries and counterexamples. *Advances in Neural Information Processing Systems*, 32, 2019.

[170] K. Wen, Y. Li, B. Liu, and A. Risteski. Transformers are uninterpretable with myopic methods: a case study with bounded dyck grammars. *Advances in Neural Information Processing Systems*, 36, 2024.

[171] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[172] J. Worrell. Exactly learning regular languages using membership and equivalence queries. https://www.cs.ox.ac.uk/people/james.worrell/DFA-learning.pdf, 2018. Lecture notes, Computational Learning Theory, Oxford.

[173] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

[174] A. C.-C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213, 1979.

[175] S. Yao, B. Peng, C. Papadimitriou, and K. Narasimhan. Self-attention networks can process bounded hierarchical languages. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3770–3785, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.292. URL https://aclanthology.org/2021.acl-long.292.

[176] M. Yau, E. Akyürek, J. Mao, J. B. Tenenbaum, S. Jegelka, and J. Andreas. Learning linear attention in polynomial time. *arXiv preprint arXiv:2410.10101*, 2024.

[177] X. Yu, N. T. Vu, and J. Kuhn. Learning the dyck language with attention-based seq2seq models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 138–146, 2019.

[178] C. Yun, S. Bhojanapalli, A. S. Rawat, S. Reddi, and S. Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

[179] Y. Zhang, Z. Wei, and M. Sun. Automata extraction from transformers. *arXiv preprint arXiv:2406.05564*, 2024.