Just read twice: closing the recall gap for recurrent language models

Simran Arora [1] **Aman Timalsina** [2] **Aaryan Singhal** [1] **Benjamin Spector** [1] **Sabri Eyuboglu** [1] **Xinyi Zhao** [1]
Ashish Rao [1] **Atri Rudra** [2] **Christopher Ré** [1]

Abstract

Recurrent large language models that compete with Transformers in language modeling perplexity are emerging at a rapid rate. These architectures use a constant amount of memory during inference. However, due to the limited memory, recurrent LMs cannot recall and use all the information in long contexts leading to brittle in-context learning (ICL) quality. The challenge is selecting what information to store versus discard. We observe the *order* in which information is shown to the LM impacts the selection difficulty. To formalize this, we show that the hardness of information recall reduces to the hardness of a problem called set disjointness (SD), a quintessential problem in communication complexity that requires a streaming algorithm (e.g., recurrent model) to decide whether inputted sets are disjoint. We empirically and theoretically show that the recurrent memory required to solve SD changes with set order, i.e., whether the smaller set appears first in-context. Our analysis suggests, to mitigate the reliance on data order, we can put information in the *right order* in-context or process prompts *non-causally*. Thus, we first propose: (1) JRT-PROMPT, where context gets repeated multiple times in the prompt, effectively showing the model *all* data orders. This gives 11.0 ± 1.3 points of improvement on average, with $11.9\times$ higher throughput than FlashAttention-2 for generation prefill. We then propose (2) JRT-RNN, which uses non-causal *prefix-linear-attention* to process prompts and provides 99% of Transformer quality at 360M params., 30B tokens and 96% at 1.3B params., 50B tokens on average across the tasks, with $19.2\times$ higher throughput for prefill than FA2.

[1]Stanford University [2]University of Buffalo. Correspondence to: Simran Arora <simarora@stanford.edu>.

Proceedings of the 2nd Efficient Systems for Foundation Models Workshop at the International Conference on Machine Learning (ICML), Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

Recent work has made rapid progress in developing fixed-memory recurrent architectures (e.g., Mamba (Gu and Dao, 2023) and RWKV (Peng et al., 2023)) that are competitive with attention in language modeling perplexity. During inference, these models are more memory efficient and asymptotically faster than the de-facto Transformer attention (Vaswani et al., 2017). However, there is no free lunch — due to their limited memory capacity, recurrent LMs cannot recall all the information provided in long-contexts, making in-context learning (ICL) quality brittle (Cho et al., 2014; Schlag et al., 2021; Arora et al., 2024). Despite matching in perplexity, we find a 2.8Bn parameter Mamba LM trained on 300Bn tokens of the Pile underperforms a 1.3Bn param. ($2.2\times$ smaller) Transformer LM trained on 50Bn tokens ($6\times$ fewer tokens) by 5 points, averaged across a suite of recall-intensive ICL tasks (Table 1).

Prior work (Arora et al., 2024) formalizes the tradeoff between an architecture's recall ability and memory consumption during inference by considering a simplified ICL setting shown below. Here, we have the "context" of key-value token pair mappings on the left and "questions" on the right for which the model should output 4, 6, 1, 2, 3:

$$\text{A 4 B 3 C 6 } \underbrace{\text{F 1}}_{\textbf{Key-Value}} \text{ E 2} \rightarrow \text{A ? C ? } \underbrace{\text{F ?}}_{\textbf{Query}} \text{ E ? B ?}$$

Unfortunately, recurrent models need $\Omega(N)$ space to solve the recall task (Arora et al., 2024). *This begs the question of whether we can rely on recurrent models that use constant $O(1)$ space for in-context learning.*

Luckily, models often do not need to remember *all* information provided in-context to excel at a task. The key challenge is *predicting* which subset of information (e.g., facts from documents, variable names from code) is useful to store in memory to support next token predictions. A long line of work focuses on improving the *selection mechanisms* or architectural inductive biases that recurrent language models use to select relevant information (e.g., LSTM (Hochreiter and Schmidhuber, 1997), decay rates (Gu and Dao, 2023; Yang et al., 2023), delta rules (Schlag et al., 2021; Munkhdalai et al., 2019)). Other works increase the recurrent state size in hardware efficient ways, traversing a quality-efficiency tradeoff space (Arora et al., 2024).

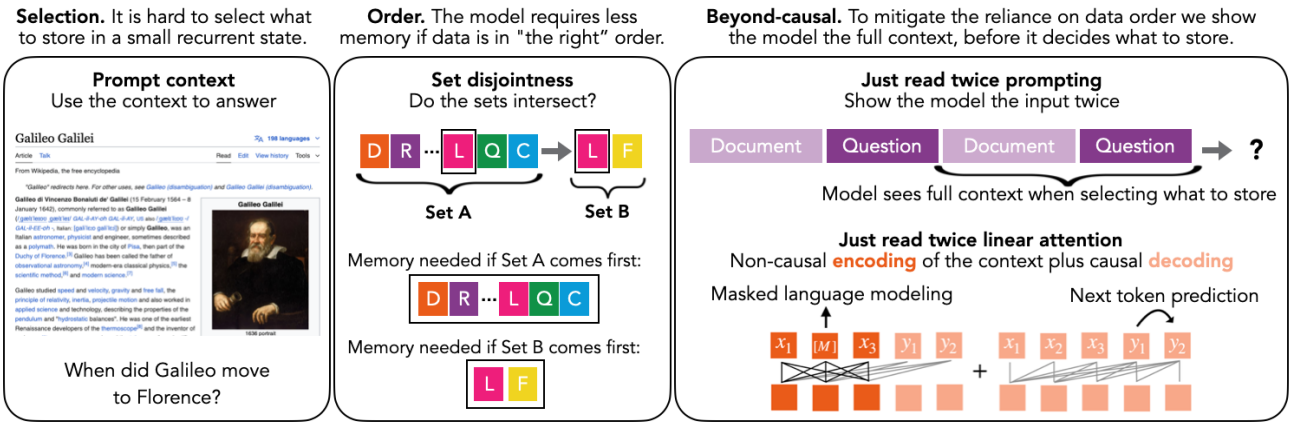

Figure 1: **Selecting (Left)** Recurrent models have limited memory and deciding what to store from long-contexts (e.g., Galileo's Wikipedia) is challenging. **Data order (Middle)** changes the selection difficulty: seeing the question *before* the document simplifies the model's selection task. We formalize this by invoking *set disjointness*, the canonical communication complexity problem of deciding whether two sets A and B are disjoint. A causal model needs enough memory to store set A to be able to compare to set B's elements so, ideally, the smaller set appears first. **Beyond causal (Right)** We show recurrent models the input twice in-context (JRT-PROMPT) or use encoder-decoder recurrent models to process the prompt (JRT-RNN), to mitigate the reliance on data order.

Complementing these approaches, we focus on the simple observation that the *order* in which data streams into the recurrent LM during inference drastically impacts the difficulty of predicting what to store in the limited memory. Suppose we ask questions $\mathcal{Q}$ (e.g., "When did Galileo move to Florence?"), over documents $\mathcal{D}$ (e.g., the Wikipedia for Galileo Galilei). The model needs to remember just one fact from $\mathcal{D}$ if the prompt is ordered $[\mathcal{Q}, \mathcal{D}]$, but needs to remember *all* facts when it is $[\mathcal{D}, \mathcal{Q}]$ (Figure 1 (Left)).

Our work first theoretically formalizes how data order impacts the memory requirement (Section 3), then proposes two ways to mitigate the reliance on data order: the Just-read-twice (JRT) prompting strategy (Section 3.2) and the JRT recurrent architecture (Section 4).

Understanding the role of data order. Our first insight is that the hardness of the recall problem reduces to the hardness of *set disjointness* (SD), the quintessential, decades-old problem in communication complexity theory (Chattopadhyay and Pitassi, 2010) (Theorem G.11). SD requires a streaming algorithm (e.g., a recurrent model) to decide whether inputted sets provided in-context are disjoint:

$$\underbrace{7\ 11\ 1\ 17\ 16\ 4\ 6\ 9}_{\textbf{Set A}} * \underbrace{8\ 1\ 5\ 6}_{\textbf{Set B}} \rightarrow \text{False}, \{1\ 6\}$$

With theory and experiments, we show that the size of the first set, $|A|$, governs the memory needed to solve SD. Causal models need to store all elements in A to be able to compare to the elements of B. This suggests that using **"the right data order"** in-context, e.g. placing the set with $\min(|A|, |B|)$ first, would help memory-limited models. Further, models that see the context **non-causally** can solve SD in space $\min(|A|, |B|)$, regardless of data order (Theorem G.15, Figure 2). We next apply these insights.

Using "the right" order. We propose JRT-PROMPT (Section 3.2), a simple strategy where information is repeated multiple times in context before the model generates answers (Figure 1 (Right)). In the second+ pass, the LM conditions on the full context when deciding what to store, effectively avoiding the issue of getting the data order "right". JRT-PROMPT gives 11.0 ± 1.3 point improvement averaged across 16 off-the-shelf recurrent LMs and the 6 ICL tasks, while providing $11.9\times$ higher throughput than FlashAttention-2 (length 32k, batch size 16) (Dao, 2024) (Table 1). JRT-PROMPT increases the context length, but remains asymptotically more efficient than attention.

Beyond causal models. We next propose JRT-RNN, inspired by the simple design of Prefix-LM encoder-decoder architectures (Raffel et al., 2020; Dong et al., 2019). Most in-context learning inputs contain two parts, the inputted prompts (context, instructions) and the text generated by the model as output. In Prefix-LMs, the LM processes the prompt region non-causally and causally decodes the output, using only a standard next token prediction loss in the causal region and in loss on the non-causal region. Unfortunately, prior approaches to training Prefix-LM models have seen limited success and use inefficient Transformer backbones (Wang et al., 2022). We apply simple changes to improve quality and efficiency including modifying the training loss and using a *linear attention* formulation we term Prefix Linear Attention (PLA). We find JRT-RNN provides a 13.7 and 6.9 point average quality improvement at 360m and 1.3b parameters, and $19.2\times$ higher throughput than FA2, using our IO-aware implementation (Table 2).

Our contributions are: (1) a synthetic and theoretical study of data order and the memory requirement for recurrent models, (2) JRT-PROMPT, and (3) JRT-

RNN. Researchers have developed many techniques for in-context leanring with Transformers (Wei et al., 2022; Creswell et al., 2022), and we need a similar exploration into how to use alternative LLM architectures effectively. Code: https://github.com/HazyResearch/prefix-linear-attention.

2. Background

We focus on developing methods for in-context learning with recurrent LLMs. We provide key background here and an extended related works discussion in Appendix A.

Recall and in-context learning. Many prior works have identified a skill called *associative recall* as highly correlated with in-context learning quality across architecture classes via extensive theoretical and empirical analysis (Graves et al., 2014; Ba et al., 2016; Schlag et al., 2021; Elhage et al., 2021; Olsson et al., 2022; Fu et al., 2023a; Arora et al., 2023a; Gu and Dao, 2023; Akyürek et al., 2024). Recall entails using information provided in context (beyond the model's memorized knowledge) to generate next token predictions. For instance, models are used via in-context learning to produce the next steps in a proof given a provided list of Lemmas (Lewkowycz et al., 2022; Trinh et al., 2024), generate the next chunk of code given a repository (Rozière et al., 2023; Yang et al., 2024), and answer questions or provide summaries given documents (Arora et al., 2023b). In a simplified view of the recall task, a model needs to remember *keys* and *values* seen in context to provide the answers for different *queries*. In this example, the model should output 4, 6, 1, 2, 3:

$$\text{A 4 B 3 C 6 } \underbrace{\text{F 1}}_{\textbf{Key-Value}} \text{ E 2} \rightarrow \text{A ? C ? } \underbrace{\text{F ?}}_{\textbf{Query}} \text{ E ? B ?}$$

Memory-recall tradeoff for causal language models. Today's LLMs process input text *causally* in a fixed left-to-right order (Brown et al., 2020). Prior work theoretically and empirically demonstrates a fundamental tradeoff between a causal LM's memory consumption during inference and its ability to remember information provided in context (recall) (Cho et al., 2014; Schlag et al., 2021; Arora et al., 2024). Attention (Vaswani et al., 2017), the de-facto LM architecture (Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023), provably solves recall perfectly in $\mathcal{O}(1)$ model depth and width as a function of sequence length. However, attention incurs $\mathcal{O}(N^2)$ complexity during training and $O(N)$ complexity and memory consumption during inference, for sequence length N. Thus, many works explore alternative recurrent architectures that are more efficient — sub-quadratic compute and memory in sequence length during training and $\mathcal{O}(1)$ during each token generation step during inference — while competing with attention in quality (Ma et al., 2022; Fu et al., 2023a; Gu and Dao, 2023; Arora et al., 2024; Yang et al., 2023, inter alia.).

However, using a limited amount memory during inference, efficient models provably cannot retain all information seen in-context, sacrificing recall and in-context learning quality (Arora et al., 2024). Models that can better select what information to store can extend the Pareto frontier of the tradeoff space. A long line of work explores how to improve this *selection mechanism* via architectural inductive biases (Hochreiter and Schmidhuber, 1997; Schlag et al., 2021; Qin et al., 2023; Gu and Dao, 2023, inter alia.). Another approach is to navigate the quality-efficiency tradeoff space by varying the recurrent *state size* in hardware-efficient ways (Massaroli et al., 2023; Arora et al., 2024; Dao and Gu, 2024). Complementing these approaches, the insight motivating our work is that the *order* in which information appears in-context drastically influences the difficulty of the selection step (Sutskever et al., 2014). *Non-causal* models can see all the input text at once to help avoid this issue.

3. Understanding the role of data order on recurrent models

In this section, we show that the quality of recurrent large language models varies as a function of the order in which data arises in context making them brittle for in-context learning applications.

3.1. Analysis of data order and communication complexity

Set disjointness problem. To formalize the impact of data order, we invoke the set disjointness (SD) problem: given two sets, determine if the intersection is empty or not. SD is the quintessential problem for studying the communication complexity of different streaming algorithms (such as recurrent models) over the past several decades (Hemaspaandra, 2010). The hardness for a wide collection of problems reduces to the hardness of SD (Chattopadhyay and Pitassi, 2010). A formal definition is in Appendix G.2.

Synthetic formulation. We construct a synthetic task where the model is given input sequences that contain two sets A and B, seperated by a special token that designates the end of set A and start of set B. Set elements are tokens $\in [0..|V|]$ for vocabulary size $|V|$ and the model needs to output the tokens in the intersection of A and B. For example, the correct output below would be 6:[1]

$$\underbrace{\text{7 11 17 16 4 6 9}}_{\textbf{Set A}} \text{ * } \underbrace{\text{8 1 5 6}}_{\textbf{Set B}} \rightarrow \text{ ?}$$

[1]Note that we train the model to output the set intersection, of size 1, not binary disjointness result (Algorithm 1). We find explicitly outputting the intersection helps the model avoid the behavior of outputting 0 or 1 with 50% accuracy during training.

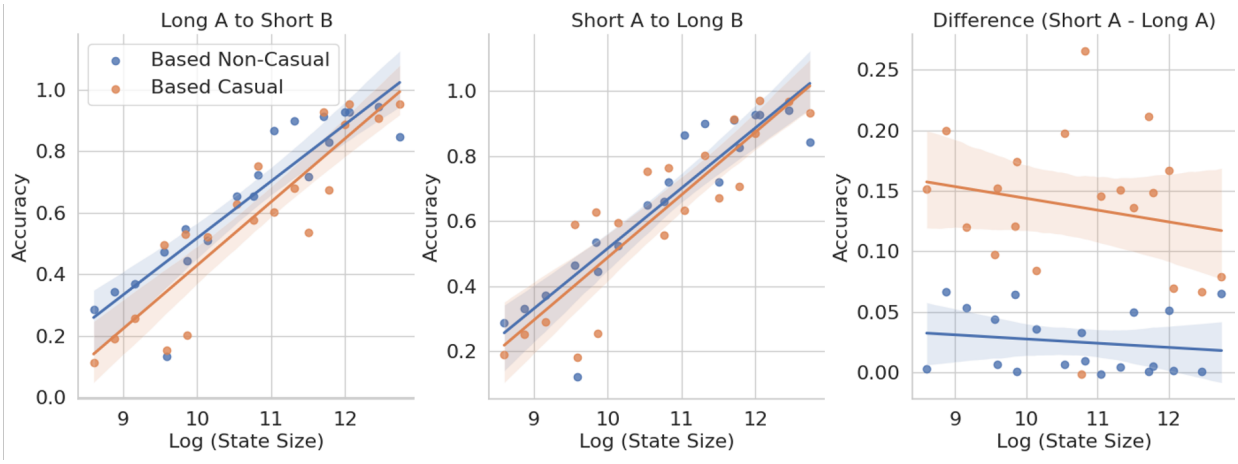

Figure 2: **Data order vs. quality.** The x-axis shows the recurrent state size in (bytes) during inference. The y-axis shows the accuracy on the set disjointness task, where the model needs to output the intersecting elements between two sets of tokens A and B (of lengths $|A|$ and $|B|$) provided in-context. **(Left)** $|A|$ is longer than $|B|$; **(Middle)** $|B|$ is longer than $|A|$; **(Right)** Difference in accuracy between the two orderings. We evaluate non-causal and causal versions of the Based recurrent architecture from (Arora et al., 2024). For each, we vary the hyperparameters (e.g., model dimension, feature dimension) that affect the state size. We plot the maximum score for each point across a sweep of three learning rates $\{1\text{e}-4, 5\text{e}-4, 8\text{e}-4\}$ and two random seeds. The plot shows that the causal recurrent models are more sensitive to the data order than non-causal models.

In Figure 2, we vary the state size of the Based recurrent architecture (Arora et al., 2024), which has been demonstrated to outperform prior subquadratic models on recall, on the SD task. We train on sequences where $|A|$ and $|B|$ are between 1 and 1024, and $|V| = 2048$. In addition to measuring overall accuracy, we consider the sliced accuracy on sequences where $|A| < |B|$ and sequences where $|B| < |A|$.

We find the causal models achieve better quality when the size of set A is smaller than set B. Figure 2 (Right) shows the difference in quality between when A is shorter vs. longer than B, reflecting that the gaps tend to be larger at smaller state sizes (x-axis). We additionally evaluate a non-causal variant of the Based architecture and find (1) it outperforms the causal models across state sizes when A is longer than B (Figure 2 (Left)), and (2) displays less variation in quality as a function of data (set) order Figure 2 (Right). We release code to reproduce this plot.

Theoretical study: recall and set disjointness. In Appendix G, we perform a systematic theoretical study of the connection between set disjointness and recall as well as the complexity of solving set disjointness in the JRT setting.

First, we show that set disjointness and the "general associative recall" (GAR) problem, which we define in Appendix G [Definition G.24]), are essentially equivalent (see Propositions G.25 and G.26). Roughly speaking, the keys and queries in GAR correspond to sets A and B in set disjointness.

We argue that recurrent models need space $\Omega(\min(|A|, |B|))$ for solving set disjointness, and hence, GAR (see Proposition G.29 in Appendix G.4.1).

Proposition 3.1. *Given a* JR$-p$ prompt[2] $\boldsymbol{u}^{\text{JR}-p} \in \{0,1\}^{pN\times d}$ *for input* $\boldsymbol{u} \in \{0,1\}^{N\times d}$ *to the* GAR *problem, any recurrent model* $\mathcal{M}_{\text{GAR}}$ *(definition G.12) solving* GAR *requires its state size to be at least* $\Omega\left(\frac{\min\{|A|,|B|\}}{p}\right)$*-bits.*

That is, the lower bound holds even if we allow multiple, but constant, many passes, as opposed to $\Omega(\max(|A|, |B|))$ lower bound for recurrent models without repeats (Arora et al., 2024) **Theorem F.3**.

Next, we show we can indeed achieve this lower bound. We show that certain recurrent models (concretely, a slight variant of `Based`) can solve SD with $O(\min(|A|, |B|))$ space in the JRT-PROMPT setting (App. G.3).

Theorem 3.2. *Given a* JRT *prompt* $\boldsymbol{u}^{\text{JRT}} \in \mathbb{R}^{2N\times(n+1)}$ *of the input* $\boldsymbol{u} \in \mathbb{R}^{N\times(n+1)}$ *for the set-disjointness (SD) problem* $(A, B) \subseteq \{0,1\}^n$*, there exists a* `Based` *model (*`BaseConv` *+ MLP +* `LinearAttention` *+ MLP)*[3] *that solves SD with space* $O(\min\{|A|, |B|\} \cdot n)$.[4]

Finally, we show that this improvement via JRT-prompting is not realizable for all possible architectures. In particular, we show that $\Omega(\max\{|A|, |B|\}) = \Omega(N)$ lower bounds for the `BaseConv` model (a model that provably simulates any gated convolution, e.g. Hyena (Poli et al., 2023a), H3 (Fu et al., 2023b), with just poly-log blowup in parameters and

[2] A JR$-p$ prompt is simply repeating the input p times (see Definition G.28).

[3] This matches the architecture in our experiments.

[4] This bound is for the case where the IP kernel is dependent on A and B; if we use an *input-independent* IP kernel, then we get an upper bound of $O\left((\min\{|A|, |B|\})^2 \cdot n\right)$ (see Remark G.23). Further, this result needs one layer of `BaseConv` where the convolution kernel is input dependent as well.

depth) (Theorems F.4, F.5, and F.6, (Arora et al., 2024)) for recall carry over even in the JRT-prompt setting (see Theorems G.6, G.7, and G.11).

3.2. Consequences of analysis on downstream in-context learning with large language models

We next show that our analysis holds consequences for in-context learning on real-world tasks.

JRT-PROMPT approach. In-context learning tasks take as input $(\mathcal{C}, \mathcal{Q}, \mathcal{Y})$ where $\mathcal{C}$ is some context (e.g., document or code repository), $\mathcal{Q}$ is some question or request to the model given the context, and $\mathcal{Y}$ is the answer. For standard in-context learning with autoregressive LM $\mathcal{A}$, we input $\mathcal{C}$ and $\mathcal{Q}$ and evaluate the generated output $\hat{\mathcal{Y}} = \mathcal{A}(\mathcal{C}, \mathcal{Q})$ against the true completion $\mathcal{Y}$.

We propose JRT-PROMPT, an exceedingly simple method in which information from the prompt (e.g. questions and documents) is *repeated* in-context before the model is prompted to output the answer, *e.g.,* $\hat{\mathcal{Y}} = \mathcal{A}(\mathcal{C}, \mathcal{Q}, \mathcal{C}, \mathcal{Q})$, as depicted in Figure 1 (Right). As a result, during the second occurrence of the context, the model can condition on a full view of the context when deciding what to store. We provide the prompts that we use in Appendix E, and release our code to reproduce the table.

Evaluation. JRT-PROMPT can be used with off-the-shelf LLMs. We evaluate the following LMs on a suite of recall-intensive in-context learning tasks, with zero-shot prompts:

- **Based** (Arora et al., 2024) pretrained LMs at the 1.3B parameter scale trained on $10 - 50$B tokens of the Pile (Gao et al., 2020). Transformer++ and Mamba models trained on the exact same tokens and data order are provided for quality references: `https://huggingface.co/collections/hazyresearch/`
- **Mamba** (Gu and Dao, 2023) pretrained LMs at the 130M, 370M, 1.4B, 2.8B parameter scales, trained on 300B tokens of the Pile (Gao et al., 2020): `https://huggingface.co/state-spaces`
- **Gated Linear Attention** (Yang et al., 2023) pretrained LMs at the 1.3B and 2.7B parameter scales, trained on 100B tokens of SlimPajama data (Computer, 2023): `https://huggingface.co/fla-hub`
- **Mamba-2** (Dao and Gu, 2024) pretrained LMs at the 130M, 370M, 1.3B, 2.7B parameter scales, trained on 300B tokens of the Pile (Gao et al., 2020): `https://huggingface.co/state-spaces`

The results are summarized in Table 1. Arora et al. (2024) finds that linear recurrent models like Mamba drastically underperform Transformers on these recall-intensive tasks. Architectures like Based increase the recurrent state size, improving both quality and efficiency, and recently Mamba-2 adopts this approach as well. Complementing the approach of increasing state size, we find the JRT-PROMPT modification provides 11.0 ± 1.3 points of improvement, averaged across models and tasks: Based models with JRT-PROMPT outperform the Transformer models with standard prompting on average. We also find that JRT-PROMPT can benefit the Transformer models and that the method appears more effective than few-shot learning for these tasks (Appendix E). Notably, Springer et al. (2024) recently proposes repeating the context for the goal of generating embeddings using autoregressive Transformer-based models, and our findings are in similar spirit. We focus on sub-quadratic architectures and in-context learning tasks.

JRT-PROMPT increases the context length due to repetition, however using using sub-quadratic recurrent architectures, this is still asymptotically more efficient than using quadratic Transformer models. We find at sequence length $N = 32768$, batch size 16, Based with JRT-PROMPT ($2N$ the sequence length) can provide $11.9\times$ higher throughput than FlashAttention-2 (N sequence length) on an NVidia H100 (see Section 5).

4. JRT-RNN: an encoder-decoder recurrent architecture

We have shown that the recall quality of causal fixed-memory recurrent models varies depending on the order in which the information appears in context, making them brittle for in-context learning. To improve reliability, we next propose a simple linear attention architecture that goes beyond causal modeling.

A long line of work has demonstrated the strength of non-causal bidirectional neural networks in language modeling (Schuster and Paliwal, 1997; Kosko, 1988; Graves and Schmidhuber, 2005; Devlin et al., 2019; Raffel et al., 2020; Patel et al., 2023). However, it is challenging to use them for fast text generation because the context must be re-processed for each generated token (Tay et al., 2023; Dong et al., 2019; Patel et al., 2023). Encoder-decoder architectures with a bidirectional encoder and causal decoder offer a way to achieve fast causal generation while reaping the benefits of bidirectional LMs. Nonetheless, decoder-only causal LMs remain the norm and encoder-decoder architectures have received little attention in the context of sub-quadratic efficient LLMs.

4.1. Preliminaries

Baseline linear recurrent architecture. We start from a recurrent architecture, linear attention, introduced in (Katharopoulos et al., 2020a; Tsai et al., 2019; Choroman-

Architecture	Params	Tokens	FDA	SWDE	NQ	SQUAD	TriviaQA	Drop	Average
Transformer++	1.3B	10B	74.4/**86.1**	41.4/**52.5**	28.2/**31.9**	39.0/**53.1**	**49.5**/49.3	22.3/**33.6**	42.5 / **51.1**
Mamba	1.3B	10B	23.3/**40.3**	15.5/**31.8**	19.4/**25.8**	26.6/**48.5**	46.4/**51.1**	21.3/**32.1**	25.1 / **38.2**
Based	1.3B	10B	48.6/**58.9**	27.6/**44.7**	19.7/**28.4**	31.0/**46.7**	44.1/**51.9**	19.5/**34.6**	31.8 / **44.2**
Transformer++	1.3B	50B	83.7/**89.2**	50.8/**65.0**	32.8/**37.5**	41.1/**58.1**	56.6/**58.8**	21.5/**37.9**	47.8 / **57.8**
Mamba	1.3B	50B	41.9/**55.7**	32.6/**45.4**	26.9/**33.9**	31.5/**53.5**	54.9/**56.7**	20.4/**33.8**	34.7 / **46.5**
Based	1.3B	50B	60.2/**68.3**	37.1/**54.0**	29.4/**35.2**	38.9/**56.3**	54.5/**57.6**	21.7/**39.1**	40.3 / **51.8**
GLA	1.3B	100B	48.3/**68.6**	37.7/**53.6**	26.6/**31.3**	34.7/**54.8**	**55.5**/54.6	19.6/**33.3**	36.7 / **48.9**
GLA	2.7B	100B	47.1/**65.8**	43.6/**54.5**	27.1/**32.9**	37.2/**55.7**	**57.9**/57.0	22.2/**34.0**	39.2/ **50.0**
Mamba	130M	300B	25.7/**32.8**	17.5/**31.5**	16.8/**21.7**	27.1/**51.9**	43.5/**50.1**	17.4/**30.7**	24.7 / **36.5**
Mamba	370M	300B	41.9/**58.3**	27.6/**42.2**	23.8/**31.1**	34.9/**51.0**	**53.6**/51.7	19.3/**33.2**	33.5 / **44.6**
Mamba	1.4B	300B	45.8/**60.9**	37.6/**46.0**	31.0/**36.6**	39.9/**59.6**	60.5/**61.3**	20.9/**36.4**	39.3 / **50.1**
Mamba	2.8B	300B	54.3/**66.6**	38.9/**48.9**	33.5/**40.1**	43.9/**59.4**	**66.2**/63.9	19.8/**36.9**	42.8 / **52.6**
Mamba-2	130M	300B	32.2/**50.9**	29.5/**43.3**	20.6/**28.9**	30.4/**47.0**	43.7/**47.2**	18.0/**34.0**	29.1 / **42.0**
Mamba-2	370M	300B	60.8/**76.7**	38.3/**52.1**	26.6/**33.6**	35.3/**51.8**	54.6/**54.7**	22.4/**36.3**	39.7 / **50.9**
Mamba-2	1.3B	300B	66.8/**74.7**	50.0/**59.6**	33.6/**40.5**	42.9/**59.6**	**63.8**/62.4	23.2/**36.6**	46.7 / **55.6**
Mamba-2	2.7B	300B	68.7/**81.6**	55.2/**60.8**	34.4/**41.7**	45.4/**59.4**	66.4/**66.5**	23.0/**42.5**	48.9 / **58.8**

Table 1: **Evaluation of pre-trained language models.** In each cell, we report in-context learning accuracy for the default zero-shot / JRT-PROMPT methods (using prompts provided in Appendix F). We evaluate across a suite of popular recall-intensive benchmarks. The zero-shot prompt includes up to 1k tokens in the input and JRT-PROMPT includes up to 2k tokens in the input for all tasks (due to repeating twice).

ski et al., 2020). Current strong recurrent LMs (e.g., Based (Arora et al., 2024), GLA (Yang et al., 2023), Mamba-2 (Dao and Gu, 2024)) adopt linear attention with large recurrent state sizes. Prior work also theoretically shows that linear attention and state space models like Mamba (Gu and Dao, 2023) are closely related (Arora et al., 2023a; 2024; Dao and Gu, 2024).

Let $\boldsymbol{q}$, $\boldsymbol{k}$, $\boldsymbol{v}$ be linear projections of the input $\boldsymbol{u} \in \mathbb{R}^{N \times d}$. The exponential in softmax attention is replaced by a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$, from model dimension d to feature dimension $\tilde{d}$, such that $\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j) \approx \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d})$. The linear attention computation can then be written as:

$$\boldsymbol{y}_i = \frac{\phi(\boldsymbol{q}_i) \sum_{j=1}^{i} \left(\phi(\boldsymbol{k}_j)^\top \boldsymbol{v}_j\right)}{\phi(\boldsymbol{q}_i) \sum_{j=1}^{i} \phi(\boldsymbol{k}_j)} \tag{1}$$

Multiplying keys and values first, the time and space complexity is $\mathcal{O}(Nd\tilde{d})$ vs. $O(N^2 d)$ for softmax attention.

Recurrent inference is split into two phases: *prefill* to process the input prompt and *decoding* to generate one token of the output at a time. During *prefill*, a length-l prompt is processed in parallel according to Equation (1) resulting in a "KV-state" $\boldsymbol{s}_l = \sum_{j=1}^{l} \phi(\boldsymbol{k}_j)^\top \boldsymbol{v}_j$ and "K-state" $\boldsymbol{z}_l = \sum_{j=1}^{l} \phi(\boldsymbol{k}_j)^\top$. During *decoding*, we can compute Equation (1) as:

$$\boldsymbol{s}_i = \boldsymbol{s}_{i-1} + \phi(\boldsymbol{k}_i)^\top \boldsymbol{v}_i, \qquad \boldsymbol{z}_i = \boldsymbol{z}_{i-1} + \phi(\boldsymbol{k}_i)^\top$$

$$\boldsymbol{y}_i = \frac{\phi(\boldsymbol{q}_i) \boldsymbol{s}_i}{\phi(\boldsymbol{q}_i) \boldsymbol{z}_i} \tag{2}$$

where $\boldsymbol{s}_i \in \mathbb{R}^{d \times \tilde{d}}$ and $\boldsymbol{z}_i \in \mathbb{R}^{\tilde{d}}$. Each decode step has $O(1)$ time and space complexity as the sequence length grows, improving upon $O(N)$ for softmax attention with KV-caching.

Prefix-LM architecture. Prefix-LM is a category of encoder-decoder models where inputs of length N are split into two regions: the first of length M is processed non-causally and the latter of length $(N - M)$ is processed causally (Raffel et al., 2020). During loss computation, the former tokens are ignored and next-token-prediction loss is computed on the latter region. Excitingly, the design is quite simple, however prior instantiations of Prefix-LMs use inefficient softmax attention backbones and have not provided compelling benefits over decoder-only Transformers (Wang et al., 2022). Prior prefix LM architectures have seen limited adoption.

4.2. JRT-RNN architecture

JRT-RNN draws inspiration from Prefix-LMs, but focuses on expanding the Pareto frontier of the quality-efficiency tradeoff space. To improve quality, JRT-RNN uses separate $\boldsymbol{k}_e$, $\boldsymbol{v}_e$ projections on the encoder side and $\boldsymbol{k}_d$, $\boldsymbol{v}_d$ projections on the decoder side. While Prefix LM models use shared projection weights for the encoder and decoder regions, we find that using two sets of projections improves quality. This observation appears in early work on recurrent encoder-decoder architectures (Sutskever et al. (Sutskever et al., 2014)).

For efficiency, JRT-RNN uses non-causal linear attention for the encoder plus standard causal linear attention for the decoder. We term this Prefix Linear Attention (PLA) (Figure 1 (Right)):

$$\boldsymbol{y}_i = \frac{\phi(\boldsymbol{q}_i)(\sum_{j=1}^{i} \phi(\boldsymbol{k}_{d_j})^\top \boldsymbol{v}_{d_j} + \sum_{j=1}^{M} \phi(\boldsymbol{k}_{e_j})^\top \boldsymbol{v}_{e_j})}{\phi(\boldsymbol{v}q_i)(\sum_{j=1}^{i} \phi(\boldsymbol{k}_{d_j})^\top + \sum_{j=1}^{M} \phi(\boldsymbol{k}_{e_j})^\top)} \tag{3}$$

Prior work has proposed many different instantiations of linear attention by varying the feature map ϕ – PLA is a general approach, agnostic to the choice of feature map.

PLA retains the linear recurrent view, $\mathcal{O}(1)$ time and space complexity for the inference decode step and the sub-quadratic in sequence length training complexity of standard causal linear attention (Katharopoulos et al., 2020b). During *prefill*, we process a length-l prompt in parallel according to Equation (3). If $l < M$, we left-pad the prefill to length M and mask the padded region during the linear attention computation. The recurrent state is initialized as:

$$\boldsymbol{s}_M = \sum_{j=1}^{M} (\phi(\boldsymbol{k}_{e_j})^\top \boldsymbol{v}_{e_j} + \phi(\boldsymbol{k}_{d_j})^\top \boldsymbol{v}_{d_j}),$$

$$\boldsymbol{z}_M = \sum_{j=1}^{M} (\phi(\boldsymbol{k}_{e_j})^\top + \phi(\boldsymbol{k}_{d_j})^\top) \tag{4}$$

Decoding for outputs $y_i, i > M$ proceeds according to Equation (2), without modification.

Efficiency. Although linear attention is theoretically more efficient than softmax attention, existing implementations are generally *slower* than well-optimized standard attention implementations (e.g., FlashAttention (Dao, 2024)). Excitingly, (Arora et al., 2024) recently provides an IO-aware kernel that realizes the efficiency benefits of the Based linear attention architecture by carefully paritioning and storing the large matrix-valued recurrent state across warp-registers during prefill (Algorithm 1 in (Arora et al., 2024)). We extend their algorithm to support PLA, using the Based feature map (defined in Appendix D) in Algorithm 2 and provide the efficiency results in Section 5. Additional details of our implementation are provided in Appendix D.

The baseline causal linear attention takes $2BNHD$ FLOPS to compute the feature map on $\boldsymbol{q}_d$, $\boldsymbol{k}_d$, and $4BNHdD$ FLOPS for the $\boldsymbol{k}_d$, $\boldsymbol{v}_d$ dot product, cumulative sum, $\boldsymbol{q}_d$ dot product, and sum along the feature dimension D respectively. PLA increases the FLOPS by $BMHD$ to compute the feature map on $\boldsymbol{k}_e$ and $3BMHdD$ to compute the $\boldsymbol{k}_e$, $\boldsymbol{v}_e$ dot product, sum along D, and sum the state with the decoder KV-state. PLA uses the same amount of memory (recurrent state size) during the inference decoding step as the original causal linear attention architecture.

4.3. JRT-RNN training objective

Our baseline recurrent models are trained with a standard next token prediction (NTP) objective, learning a probability distribution $\mathrm{P}(u_{i+1}|\{u_1, ..., u_i\})$ from input sequences of tokens $\mathbf{u} = \{u_1, ..., u_N\}$ for sequence length N, and cross-entropy loss. For the pure decoder models, the loss ($\mathcal{L}_{\text{NTP}}$) is computed using all N tokens in $\mathbf{u}$. Prefix-LMs can only compute the NTP loss ($\mathcal{L}_{\text{NTP}}$) for tokens $\{u_M, ..., u_N\}$, which are processed causally.

Prefix LMs typically compute no loss on the non-causal region, however in JRT-RNN, we combine next token prediction with the masked language modeling (MLM) objective (Devlin et al., 2019). For the added MLM objective, we replace proportion P of of tokens from the encoder region $\{u_1, ..., u_M\}$ with a [MASK] token and we measure the cross-entropy loss ($\mathcal{L}_{\text{MLM}}$) in predicting the original token. The loss is:

$$\mathcal{L} = \frac{w_1 \mathcal{L}_{\text{NTP}} + w_2 \mathcal{L}_{\text{MLM}}}{w_1 + w_2} \tag{5}$$

where $w_1, w_2 \in \mathbb{R}$ are scalar weights. During inference, no [MASK] tokens are used; inference proceeds as with causal LMs.

5. Results

In this section, we validate the following quality and efficiency claims for JRT-RNN:

1. **In-context learning (ICL) quality** JRT-RNN provides 99% of Transformer quality at 360M params./30Bn tokens, averaged across the recall-intensive ICL benchmarks. This represents 46.7% improvement over Based and 78.8% over Mamba. JRT-RNN provides 96% of Transformer quality at 1.3Bn params./ 50Bn tokens, representing 16.2% improvement over Based and 34.5% over Mamba on average.
2. **Overall language modeling** Beyond outperforming in recall, we show that JRT-RNN matches the baselines in general natural language understanding (SuperGLUE). We give a detailed analysis of the pretrained LMs, comparing perplexity on slices of the Pile test set to show the strengths and limitations.
3. **Generation** We show that JRT-RNN can provide 19.2× higher prefill throughput than FlashAttention-2 at 32k sequence length, batch size 16 on an NVidia H100 GPU.

Models. We compare JRT-RNN to two state-of-the-art recurrent autoregressive models, Based (Arora et al., 2024) and Mamba (Gu and Dao, 2023). We also compare to the Transformer++ (Llama architecture (Touvron et al., 2023)), which adds rotary encodings (Su et al., 2023) and GLU.

For JRT-RNN, we start from the Based linear recurrent architecture, since it has been shown in prior work to outperform prior sub-quadratic architectures (e.g., Mamba, GLA) at recall. An extended explanation of Based is in Appendix D. We reiterate that the approaches in JRT-PROMPT and JRT-RNN can be combined with any recurrent LM.

Benchmarks. We evaluate on a range of ICL benchmarks. We use SuperGLUE to test general language understanding (Wang et al., 2019). We next evaluate on a suite of recall-intensive tasks including: SWDE and FDA information extraction tasks (Wu et al., 2021; Deng et al., 2022;

Architecture	Param/Tok	FDA 512 Acc ↑	FDA 1024 Acc ↑	SWDE 512 Acc ↑	SWDE 1024 Acc ↑	NQ 512 Acc ↑	NQ 1024 Acc ↑	SQUAD Full Acc ↑	Trivia Full Acc ↑	Drop Full Acc ↑	Avg. Acc ↑
Transformer	360M/30B	74.8	**73.0**	**44.7**	**43.0**	27.8	**22.9**	36.2	**46.5**	21.8	**43.4**
Mamba	360M/30B	41.1	24.3	22.2	13.6	16.4	12.5	25.5	43.0	17.3	24.0
Based	360M/30B	50.3	35.8	30.4	21.6	19.7	14.7	29.8	42.5	18.4	29.2
JRT-RNN	360M/30B	**82.0**	66.0	43.3	35.1	**32.9**	16.2	**41.7**	43.2	**25.8**	42.9
Transformer	1.3B/10B	75.3	**71.5**	**41.6**	**41.0**	**29.6**	**25.8**	38.7	**48.8**	22.6	**43.9**
Mamba	1.3B/10B	37.4	23.3	23.0	15.1	19.6	16.1	26.1	45.7	20.9	25.2
Based	1.3B/10B	66.3	49.0	32.3	26.3	19.7	15.7	30.7	44.2	19.1	33.7
JRT-RNN	1.3B/10B	**78.5**	60.6	38.5	32.7	26.5	16.7	**51.6**	44.8	**28.4**	42.0
Transformer	1.3B/50B	85.6	**83.5**	**55.7**	**56.0**	33.4	**29.9**	40.1	**56.6**	21.4	**51.4**
Mamba	1.3B/50B	55.4	40.1	44.0	33.7	27.6	23.2	32.2	54.5	20.7	36.8
Based	1.3B/50B	69.3	58.8	47.6	40.4	29.1	24.4	38.5	54.3	20.8	42.6
JRT-RNN	1.3B/50B	**86.7**	67.7	49.4	45.7	**38.3**	25.4	**50.4**	53.0	**29.3**	49.5

Table 2: **Evaluation of JRT-RNN models.** We compare JRT-RNN to strong LMs proposed in prior work (Based, Mamba, and Transformer++) across parameter scales. In the table, we specify the length (number of tokens) of the documents provided in context (512, 1024, Full), where "Full" means the full document is included as prefill. Table 7 contains the average number of tokens per document in each benchmark.

Arora et al., 2023b; 2024), where the model needs to extract values for a specified attribute from in-context passages, and SQUADv2 (Rajpurkar et al., 2018), Natural Questions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and Drop (Dua et al., 2019). In these tasks, the model needs to ground its answers in in-context documents. We release code and models to reproduce our results and provide details on the benchmarks and evaluations in Appendix B.

5.1. In-context learning quality

In Table 2, we find JRT-RNN outperforms the decoder-only baseline (Based) by 13.7 points at 360M parameters (30Bn tokens) and 6.9 points at 1.3B parameters (50Bn tokens) on average. JRT-RNN closes the gap to Transformer++ to within 0.5 points on average at 360M and 1.9 points on average at 1.3B parameters.

In Table 2, we left pad documents with length $< M$, where $M = 1024$ is the encoder region's length during training (discussed in Section 4) – for the three results with length 512 documents we pad using JRT-PROMPT and otherwise with the tokenizer's space token (discussed further below).

Length extrapolation. Though the encoder processes until length $M = 1024$ for our trained LMs, we excitingly find that the benefits of JRT extend to prefill lengths l s.t. $l > M$ as well. In Section 5.1, we evaluate at the 360M and 1.3B parameter scales with documents of length 2000.

Inference strategies. In Table 3, we compare alternate inference strategies for JRT-RNN in the regime where the prefill length l is less than the encoder length M, $l < M$:

- **Decoding with padding**: We left-pad the prefill to length M to match the training distribution the model sees. Causal decoding starts at position M. This is the default for JRT-RNN.
- **Read-twice pad**: Instead of padding with a special token, we can "pad" by repeating the context (i.e., JRT-PROMPT). We use this at $l = 512$ for FDA, SWDE, and NQ in Table 2. Padding is a fixed cost for JRT-RNN, so it can be used creatively.
- **Iterative encoding**: We allow the model to *non-causally view its previously generated tokens* during decoding. We generate token $\boldsymbol{y}_l$ given the length l prefill, append it to the prefill, and then compute $\boldsymbol{y}_{l+1}$ again using the parallel view on the new input of length $l + 1$. This protocol is expensive, but future work could consider *periodically* updating the non-causal encoder-state when decoding many tokens.

Arch.	Param/Tokens	FDA 2k	SWDE 2k	NQ 2k
Transformer	360M/10B	65.2	41.0	23.0
Mamba	360M/10B	12.4	13.4	12.4
Based	360M/10B	19.1	18.9	13.9
JRT-RNN	360M/10B	28.4	26.1	15.4
Transformer	1.3B/50B	79.7	55.5	30.2
Mamba	1.3B/50B	21.0	29.9	23.1
Based	1.3B/50B	36.1	37.7	23.4
JRT-RNN	1.3B/50B	55.2	41.4	26.2

Table 3: Evaluation at prefill lengths 2k, i.e. beyond the encoder region (length $M = 1024$).

Inference	Param/Tokens	FDA 512	SWDE 512	NQ 512
Left-pad	360M/30B	61.9	38.1	24.6
Read-2×	360M/30B	82.0	43.3	32.9
Iterate	360M/30B	76.3	40.7	29.2
Left-pad	1.3B/50B	75.8	49.3	30.9
Read-2×	1.3B/50B	86.7	49.4	38.3
Iterate	1.3B/50B	80.2	43.3	34.2

Table 4: JRT-RNN with alternate inference strategies when $l < M$, for prefill and encoder lengths l and M.

Overall natural language understanding While recall is critical for in-context learning, it is important to validate that the models remain strong in their overall natural language understanding abilities.

In appendix C we provide a detailed analysis of the perplexity of each architecture on various slices of the Pile test set, showing that JRT-RNN helps predict tokens that require in-context recall. We also use the downstream SuperGLUE benchmark, a canonical test of natural language understanding ability (Wang et al., 2019), to evaluate each architecture

at the 360M and 1.3B parameter scales in Table 8. We validate that the different architectures perform similarly on average across these generic, short-context language tasks as observed in prior work (Fu et al., 2023c; Karami and Ghodsi, 2024; Arora et al., 2024).

Table 5: Latency (ms) of inference prefill for each implementation. Each point is the average of 20 iterations, run on an NVIDIA H100 GPU. In Table 5, we vary the sequence length at a fixed batch size of 16. In Table 5, we vary the batch size at a fixed sequence length of 16384.

Implementation	2048	4096	8192	16384	32768
Based PyTorch	17.1	74.5	284.6	OOM	OOM
Fast Transformer CUDA	11.4	23.0	47.0	96.0	OOM
Based Triton (FLA)	1.0	2.8	9.3	32.6	123.7
Based Custom CUDA	0.3	0.6	1.2	2.3	4.5
FlashAttention-2	0.5	1.8	6.8	26.6	107.8
JRT-RNN PyTorch	21.3	89.2	OOM	OOM	OOM
JRT-PROMPT Custom CUDA	0.6	1.2	2.3	4.5	9.0
JRT-RNN Custom CUDA	0.4	0.8	1.5	2.8	5.6

Implementation	8	16	32	64
Based PyTorch	OOM	OOM	OOM	OOM
Based Triton (FLA)	16.7	32.4	64.2	127.8
Based Custom CUDA	1.5	2.3	4.5	8.9
FlashAttention-2	13.4	26.6	52.9	108.2
Fast Transformer CUDA	50.7	95.5	OOM	OOM
JRT-RNN PyTorch	OOM	OOM	OOM	OOM
JRT-PROMPT Custom CUDA	2.9	4.5	9.0	17.8
JRT-RNN Custom CUDA	1.8	2.8	5.6	11.1

5.2. Generation throughput

Generation can be decomposed into prompt "prefill processing" and decoding "next token prediction" steps. Since JRT-RNN does not modify the decoding step relative to standard decoder-only recurrent models, we focus our discussion on the prefill stage.

Using the Based CUDA kernel proposed in (Arora et al., 2024), JRT-PROMPT gives $11.9\times$ and $13.7\times$ higher throughput in processing the prompt prefill than the FlashAttention-2 and FLA Triton kernels respectively (prefill length 32768) (Table 5). JRT-PROMPT provides $6.1\times$ and $7.2\times$ higher throughput than the FlashAttention-2 and FLA kernels respectively as we increase the batch size to 64 (Table 5). For JRT-PROMPT, we *double the prefill length* compared to the baselines, using $2\times$ the time of the original Based prefill.

We next extend the Based kernel to support JRT-RNN and demonstrate that the implementation achieves $19.2\times$ and $22.0\times$ higher throughput than FA2 and FLA as we increase sequence length to 32768 (Table 5). JRT-RNN provides $9.7\times$ and $11.5\times$ higher throughput respectively as we increase the batch size to 64 (Table 5). JRT-RNN takes $1.24\times$ the time of the Based prefill.

We benchmark the inference efficiency of JRT-PROMPT and JRT-RNN in Table 5 (additional details in Appendix D). As baselines, we consider popular and well-optimized softmax attention and linear attention implementation. For attention, we consider FlashAttention-2 (Dao, 2024). For linear attention, we consider the linear attention CUDA kernel from Fast Transformers (Katharopoulos et al., 2020b; Vyas et al., 2020) and a Triton parallel Based kernel from Flash Linear Attention (FLA) (Yang and Zhang, 2024). We also compare to PyTorch implementations of JRT-RNN and Based. All numbers are benchmarked on a NVidia H100 GPU.

6. Conclusion

Recurrent LLMs promise drastically more efficient inference relative to Transformers, however they are brittle during in-context learning. We identify the role of data order as a key reason, formalized via synthetics and theory. Our analysis suggest that putting data in the right order in context or non-causally processing the context can help efficient recurrent models better use their limited memory. We translate these insights to JRT-PROMPT and JRT-RNN respectively. JRT-PROMPT improves the quality of recurrent models by 11.0 ± 1.3 points averaged across models and tasks, and our prototype architecture, JRT-RNN, provides a 13.7 point improvement at 360M parameters and 6.9 point improvement at 1.3B parameters. Both methods increase throughput relative to FlashAttention-2 using IO-aware CUDA implementations.

While much of the effort on sub-quadratic LMs seeks to directly mimic the experience of using quadratic Transformer LMs, our work emphasizes that we can exploit the asymmetries in efficiency to close the quality gaps: *multiple* linear passes over data is still asymptotically more efficient than quadratic attention. To facilitate reproducing this work, we release code and models at `https://github.com/HazyResearch/prefix-linear-attention`.

Acknowledgments

We thank Michael Zhang, Michael Poli, Daniel Fu, Kawin Ethayarajh, John Thickstun, and Neel Guha for their helpful feedback and discussion during this work. We thank the Hazy Research lab and Together AI for supporting this work. We gratefully acknowledge the support of NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF2247015 (Hardware-Aware), CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), and 1937301 (RTML); US DEVCOM ARL under Nos. W911NF-23-2-0184 (Long-context) and W911NF-21-2-0251 (Interactive Human-AI Teaming); ONR under Nos. N000142312633 (Deep Signal Processing), N000141712266 (Unifying Weak Supervision), N000142012480 (Non-Euclidean Geometry), and N000142012275 (NEPTUNE); Stanford HAI under No. 247183; NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft,

NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, Google Cloud, Salesforce, Total, the HAI-GCP Cloud Credits for Research program, the Stanford Data Science Initiative (SDSI), and members of the Stanford DAWN project: Facebook, Google, and VMWare. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of NIH, ONR, or the U.S. Government. AR's research is supported by NSF grant CCF#2247014.

References

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, and Jiaming et al. Kong. Rwkv: Reinventing rnns for the transformer era. *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR, 2021.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *International Conference on Machine Learning*, 2024.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation 9*, 1997.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *International Conference on Machine Learning*, 2023.

Tsendsuren Munkhdalai, Alessandro Sordoni, Tong Wang, and Adam Trischlern. Metalearned neural memory. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

Arkadev Chattopadhyay and Toniann Pitassi. The story of set disjointness. *ACM SIGACT News*, 41(3):59–85, 2010.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *International Conference on Learning Representations*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. What language model architecture and pretraining objective work best for zero-shot generalization? *Proceedings of the 39 th International Conference on Machine Learning*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022.

Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. *International Conference on Machine Learning (ICML)*, 2022.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1, 2021.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry Hungry Hippos: Towards language modeling with state space models. In *International Conference on Learning Representations*, 2023a.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *The Eleventh International Conference on Learning Representations*, 2023a.

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms. *International Conference on Machine Learning*, 2024.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 2024.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2023. URL https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code/.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv:2405.15793*, 2024.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proceedings of the VLDB Endowment*, 2023b.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, and Shruti Bhosale. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*, 2023.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Zettlemoyer Luke. Mega: Moving average equipped gated attention. *International Conference on Learning Representations (ICLR)*, 2022.

Zhen Qin, Songlin Yang, and Yiran Zhong. Hierarchically gated recurrent neural network for sequence modeling. *Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.

Stefano Massaroli, Michael Poli, Daniel Y Fu, Hermann Kumbong, David Romero, Rom Parnichukun, Aman Timalsina, Quinn McIntyre, Beidi Chen, Atri Rudra, Ce Zhang, Christopher Ré, Stefano Ermon, and Yoshua Bengio. Laughing hyena distillery: Extracting compact recurrences from convolutions. *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *International Conference on Machine Learning (ICML)*, 2024.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

Lane A. Hemaspaandra. Sigact news complexity theory column 67. *ACM SIGACT News*, 41, 2010.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards

larger convolutional language models. *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023a.

Daniel Y. Fu, Elliot L. Epstein, Eric Nguyen, Armin W. Thomas, Michael Zhang, Tri Dao, Atri Rudra, and Christopher Ré. Simple hardware-efficient long convolutions for sequence modeling. *Proceedings of the 40 th International Conference on Machine Learning (ICML)*, 2023b.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. Repetition improves language model embeddings. *arXiv:2402.15449*, 2024.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. In *IEEE Transactions on Signal Processing*, volume 45, 1997.

Bart Kosko. Bidirectional associative memories. In *IEEE Transactions on Systems, Man, and Cybernetics*, 1988.

Alex Graves and Jurgen Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. *Proceedings of International Joint Conference on Neural Networks*, 2005.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT 2019*, 2019.

Ajay Patel, Bryan Li, Mohammad Sadegh Rasooli, Noah Constant, Colin Raffel, and Chris Callison-Burch. Bidirectional language models are also few-shot learners. *International Conference on Learning Representations (ICLR)*, 2023.

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. Ul2: Unifying language learning paradigms. *International Conference on Learning Representations (ICLR)*, 2023.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020a.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *International Conference on Learning Representations (ICLR)*, 2020.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020b. URL https://arxiv.org/abs/2006.16236.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *SuperGLUE: a stickier benchmark for general-purpose language understanding systems*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Eric Wu, Kevin Wu, Roxana Daneshjou, David Ouyang, Daniel Ho, and James Zou. How medical ai devices are evaluated: limitations and recommendations from an analysis of fda approvals. *Nature Medicine*, 27:1–3, 04 2021.

Xiang Deng, Prashant Shiralkar, Colin Lockard, Binxuan Huang, and Huan Sun. Dom-lm: Learning generalizable representations for html documents. 2022.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *ACL*, 2018.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of*

the Association for Computational Linguistics, 7:452–466, 2019. doi:10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proc. of NAACL*, 2019.

Daniel Y. Fu, Simran Arora, Jessica Grogan, Isys Johnson, Sabri Eyuboglu, Armin W. Thomas, Benjamin Spector, Michael Poli, Atri Rudra, and Christopher Ré. Monarch mixer: A simple sub-quadratic gemm-based architecture. *37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023c.

Mahdi Karami and Ali Ghodsi. Orchid: Flexible and data-dependent convolution for sequence modeling. *ICLR 2024 Workshop on Understanding of Foundation Models (ME-FoMo)*, 2024.

A. Vyas, A. Katharopoulos, and F. Fleuret. Fast transformers with clustered attention. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/sustcsonglin/flash-linear-attention.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models, 2024.

Michael Poli, Jue Wang, Stefano Massaroli, Jeffrey Quesnelle, Ryan Carlow, Eric Nguyen, and Armin Thomas. StripedHyena: Moving Beyond Transformers with Hybrid Signal Processing Models. 12 2023b. doi:10.57967/hf/1595. URL https://github.com/togethercomputer/stripedhyena.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Yair Schiff, Chia-Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. Caduceus: Bi-directional equivariant long-range dna sequence modeling. *arXiv preprint arXiv:2403.03234*, 2024.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*, 2016.

Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. *Association for Computational Linguistics (ACL)*, 2024.

Saleh Soltan, Shankar Ananthakrishnan, Jack FitzGerald, Rahul Gupta, Wael Hamza, Haidar Khan, Charith Peris, Stephen Rawls, Andy Rosenbaum, Anna Rumshisky, Chandana Satya Prakash, Mukund Sridhar, Fabian Triefenbach, Apurv Verma, Gokhan Tur, and Prem Natarajan. Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model, 2022.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. GLM: General language model pretraining with autoregressive blank infilling. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.acl-long.26.

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. *International Conference on Learning Representations (ICLR)*, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. Zeroshotceres: Zero-shot relation extraction from semi-structured webpages. *ACL*, 2020.

Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. Ask me anything: A simple strategy for prompting language models. *International Conference on Learning Representations (ICLR)*, 2022.

Thathachar S Jayram, Ravi Kumar, and Dandapani Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at http://cse. buffalo. edu/faculty/atri/courses/coding-theory/book*, 2019.

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention approximation. *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.

Johan Håstad and Avi Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, 3(1):211–219, 2007.

The appendix is organized as follows:

1. Appendix A includes an extended related works discussion.
2. Appendix B includes additional experimental details.
3. Appendix C includes additional experiments to supplement Section 5.
4. Appendix D includes details on the IO-aware implementation and benchmarking for JRT-RNN.
5. Appendix E includes error analysis discussion for JRT-PROMPT.
6. Appendix F includes the prompts used for all in-context learning experiments in this work.
7. Appendix G includes theoretical results and proofs.

A. Extended related work discussion

The notion that causal models are limited because they need to "predict the future" when computing representations is well-known (Raffel et al., 2020; Schuster and Paliwal, 1997; Kosko, 1988). Yet, current large language models (e.g., Llama (Touvron et al., 2023), GPT (Brown et al., 2020), and efficient Mamba (Gu and Dao, 2023), Griffin (De et al., 2024), GLA (Yang et al., 2023), RWKV (Peng et al., 2023), Striped Hyena (Poli et al., 2023b)) are causal. Here we provide an extended discussion of the related work.

A.1. Prompting strategies

Most related to our work, Springer et al. (2024) recently proposes to produce embeddings from autoregressive Transformer models by repeating the context twice and taking embeddings from the activations of second occurrence. We focus on 1) sub-quadratic models / memory perspective, 2) recall-intensive tasks rather than producing embeddings. Our findings build on these ideas and the key distinctions are: (1) our focus on *sub-quadratic architectures*, which can provide asymptotically higher efficiency, (2) our focus on recall and in-context learning based tasks as opposed to embedding generation, and (3) our theoretical analysis on why JRT-PROMPT impacts the memory requirement of recurrent LMs.

We are certainly not the first to try modifying the data order for recurrent LMs. The seminal Seq2seq paper from Sutskever et al. (Sutskever et al., 2014) proposes to *reverse* the order of the tokens in the source sequence when using encoder-decoder LSTM-based recurrent language models.

A.2. Encoder-decoder language models

A long line of work has explored the use of bidirectional networks (Schuster and Paliwal, 1997; Kosko, 1988; Graves and Schmidhuber, 2005; Devlin et al., 2019; Raffel et al., 2020; Patel et al., 2023). In early work, Schuster and Paliwal (1997) demonstrate synthetic math tasks that require recurrent models to use lagging and future values to produce outputs, favoring bidirectional networks. Kosko (1988) explores associative recall style tasks in two layer bidirectional networks. We build on the ideas from this line of work and focus on our discussion on large language modeling architectures.

Three popular language modeling architecture paradigms are encoder-only, decoder-only, or encoder-decoder. A popular use case for bidirectional, encoder-only, models is producing word or context embeddings (Peters et al., 2018; Devlin et al., 2019). It is challenging to use these models for fast and open-ended generation (Tay et al., 2023; Dong et al., 2019). Encoder-decoder models have emerged as a compelling alternative, combining non-causal bidirectional encoding for parts of the input text and causal decoding to generate responses.

However, causal decoder-only language models currently prevail (e.g., Llama-3 (AI@Meta, 2024), GPT (Ouyang et al., 2022; Brown et al., 2020), PaLM (Chowdhery et al., 2022)). Current research on efficient architectures also largely focuses on pure encoder-only (e.g. M2-BERT (Fu et al., 2023c), Mamba-Caduceus (Schiff et al., 2024), Orchid (Karami and Ghodsi, 2024)) or decoder-only causal LMs (e.g., Mamba (Gu and Dao, 2023), RWKV (Peng et al., 2023), Griffin (De et al., 2024), Striped Hyena (Poli et al., 2023b)), as opposed to encoder-decoder. In contrast, our work on JRT-RNN explores encoder-decoder recurrent LMs in light of recent progress in sub-quadratic efficient architectures.

Recurrent encoder-decoder language models Recurrent encoder-decoder language models were popular in the context of machine translation systems. Sutskever et al. (2014) uses two LSTM RNNs, one to process the inputs and produce a fixed dimensional vector, and the other to decode the outputs from this vector. Wu et al. (2016) use a similar two-stack (encoder-stack and decoder-stack) architecture, using right-to-left and left-to-right RNNs for some encoder layers).

Instead of compressing the source sentence into a fixed recurrent state, Bahdanau et al. (2016) use *attention* to refer back to encoder states. A key motivating observation for the switch to attention comes from Cho et al. (2014), which finds that the quality of RNN-based encoder-decoder language models degrades quickly as the sequence length increases. Following the rise of attention and the Transformer architecture (Vaswani et al., 2017) in popularity, subsequent work predominantly explores Transformer-based encoder-decoder LMs.

Transformer-based encoder-decoder language models Raffel et al. (2020) propose the T5 architecture, which uses two separate Transformer stacks, one for non-causally encoding input text and one for causally decoding response. Cross-attention allows the decoder attention queries to attend to the final attention key and value states form the encoder stack. More

recently, (Yen et al., 2024) trains a 7Bn parameter two-stack encoder-decoder model called CEPE, adapted form Llama-2 (Touvron et al., 2023) with cross-attention between stacks, following T5.[5] We evaluate this model on the recall-intensive tasks and surprisingly find that ignoring its encoder altogether and placing documents and questions in the decoder far outperforms placing the document in the encoder and questions in the decoder on the recall-intensive benchmarks.

	SWDE Acc. ↑	FDA Acc. ↑
CEPE Enc.-Dec.	51.0	5.9
CEPE Dec.-Only	80.4	72.5

Table 6: Evaluating the CEPE 7Bn parameter model (Yen et al., 2024) on the document information extraction tasks, using $N = 50$ random examples. For the encoder-decoder baseline, the document is inputted to the encoder and the question (i.e., name of the attribute to extract from the document) is sent to the decoder. In the decoder-only model, the standard prompt containing the document plus attribute are inputted to the decoder and the model's encoders are ignored (empty inputs). We observe the encoder-decoder model tends to produce irrelevant responses.

Prior work suggests that the T5 architecture struggles in open-ended generation (Patel et al., 2023; Tay et al., 2023). Some differences between JRT-RNN and the T5-style approach are that the T5 corruption pretraining objective deviates from how the models are used for downstream generation tasks, and training requires the use of multiple special sentinel tokens and unique positional encodings per stack of layers.

Instead of using separate encoder and decoder stacks, some prior work explores the use of Prefix-LMs. These models split the input into encoder and decoder regions *within* each layer, where the former is processed non-causally and the latter is processed causally (Raffel et al., 2020). Next token prediction loss is computed on the causal tokens and no loss is computed on the prefix tokens.

To better equip encoder-decoders with generation abilities, UniLM (Dong et al., 2019), UL2 (Tay et al., 2023), AlexaTM (Soltan et al., 2022) and others use different combinations of span corruption and prefix language modeling pretraining objectives. During training, given an input sequence, one of the suite of objectives is sampled with some pre-defined probability. Each of these architectures are Transformer-based, facing quadratic scaling in sequence length during training and linear scaling during inference. In GLM (Du et al., 2022), spans of text are masked and autoregressively in-filled during training, to endow the model with generation capabilities. We are inspired by these works in combining MLM and next token prediction objectives, and future work could explore alternate variations to the training objective used in JRT-RNN.

Discussing the differences in JRT-RNN Recent work has made exciting progress in designing efficient LMs that extend the Pareto-frontier of the quality-efficiency tradeoff space relative to Transformers and prior recurrent architectures. However, these are decoder-only LMs, while JRT-RNN uses the encoder-decoder framework. Prior popular encoder-decoder LMs are Transformer-based with quadratic scaling and do not convincingly improve in quality over decoder-only models (Wang et al., 2022), so the motivation to use them is unclear. JRT-RNN improves efficiency (Table 5) and quality (Table 2).

Within the encoder-decoder framework, JRT-RNN uses a prefix LM structure. Unfortunately, prior work and our ablations suggest this training strategy does not perform well ((Wang et al., 2022) and Table 11), and this architecture has not seen adoption. Instead JRT-RNN deviates by (1) adding a masked language modeling loss to the prefix alongside next token prediction for the suffix. JRT-RNN (2) *reads the prefix twice*. Prefix LM models modify the attention mask of standard attention to make the prefix non-causal and use shared projection weights for the non-causal encoder and causal decoder regions. Instead, JRT-RNN uses two sets of key and value representations for encoding and decoding respectively.

[5] https://huggingface.co/hyen/CEPED-LLaMA-2-Chat-7B

B. Experimental details

This section provides additional details for the synthetic, JRT-PROMPT and JRT-RNN experimental protocols. We use NVidia A100-80GB GPUs for all training runs.

B.1. Additional details for set disjointness synthetic experiments

This section provides experimental details for Figure 2.

Dataset The procedure for generating training and evaluation data for our synthetic experiments is shown in Algorithm 1. We train on the following mixture of sequence lengths, where the tuple denotes $(|A|, |B|)$ for sets A and B in the sequence:

$$(4, 16), (16, 4), (8, 32), (32, 8), (64, 16), (16, 64), (4, 128), (128, 4), (16, 256), (256, 16), (4, 256), (256, 4)$$

We evaluate on the following mixture of sequence lengths (requiring length extrapolation from training), where the tuple denotes $(|A|, |B|)$ for sets A and B in the sequence:

$$(1, 32), (32, 1), (4, 32), (32, 4), (4, 128), (128, 4), (16, 256), (256, 16), (4, 256), (256, 4), (16, 512),$$

$$(512, 16), (4, 512), (512, 4), (8, 768), (768, 8), (16, 768), (768, 16), (4, 768), (768, 4)$$

We include 20000 data points per tuple above during training and 1000 during evaluation. We use $V = 2048$ as the vocabulary size.

Algorithm 1 Set Disjointness Synthetic Procedure

Require: Vocabulary V, Sequence lengths N_A and N_B for sets A and B, Special token IDs prefix_token_id, mask_tok_id, sep_sets_token_id, sep_answer_tok_id

Output: Synthetic sequence

1: Let the first half of V, V_A, be prospective tokens for set A and the second half, V_B, be prospective tokens for set B.
2: Randomly select N_A tokens from V_A for set A. Randomly select N_B tokens from V_B for set B.
3: Randomly select a token t from A as the intersecting token between sets. Replace a random token (at a random position) from B with t.
4: Construct the final input sequence as the concatenation:

$$[\text{prefix_token_id}], A, [\text{sep_sets_token_id}], B, \text{sep_answer_tok_id}], [t]$$

5: The label sequence contains a "-100" (i.e., a token to ignore computing the loss) at all positions except for the final position. We mask $[t]$ (the final position) from the input sequence.
6: Output the synthetic input and label sequences.

Models We evaluate causal and non-causal variants of the Based recurrent model. Each model contains 4 layers alternating gated-convolutions (with a short filter of size 3) and linear attention with 2 query key and value heads. For the non-causal variant, we simply replace the causal cumulative sum in linear attention with a sum, and we use non-causal circular convolutions. For the linear attention feature map, we use a Taylor approximation to the softmax-exponential function as in (Arora et al., 2024) (also defined in **??**). Each layer has an MLP with GeLU activations. We do not use any explicit positional embeddings, instead finding the short-convolutions sufficient for positional information.

To sweep the state size, we vary the model width or dimension $\in \{36, 48, 64, 96, 128\}$ and linear attention feature dimension $\in \{4, 8, 16, 24\}$.

Training We train using cross-entropy loss on the predicted vs. true intersection token t in Algorithm 1. For each point in Figure 2, we sweep learning rates $\in \{0.0001, 0.0005, 0.0008\}$ (after identifying that this regime is most effective for the architectures) and report the maximum accuracy after 48 epochs of training. We use AdamW as the optimizer with 0.1 weight decay.

We build our synthetic experiments using the synthetics repository provided by prior work (Arora et al., 2023a): `https://github.com/HazyResearch/zoology`.

B.2. Additional details for JRT-PROMPT experiments

For Table 1 (JRT-PROMPT), we use the following publicly available models pretrained and released by the baseline works:

- Based (Arora et al., 2024) models are at `https://huggingface.co/collections/hazyresearch/based-65d77fb76f9c813c8b94339c`
- Gated Linear Attention (Yang et al., 2023) models are at `https://huggingface.co/fla-hub`.
- Mamba (Gu and Dao, 2023) and Mamba-2 (Dao and Gu, 2024) models are at `https://huggingface.co/state-spaces`

We integrate all tasks into the popular LM-Eval harness to run inference. We truncate long-documents (e.g., in NQ, FDA, SWDE) to length 1k tokens for the default prompting and length 2k tokens for JRT-PROMPT so that both methods receive the same information in-context. We note that these lengths are chosen because the listed pretrained models have 2048 context lengths. We ensure that the answer span is present in truncated documents. We do not use any task-specific prompt customization in this section, to highlight the effectiveness of JRT-PROMPT despite little effort.

B.3. Additional details for pre-training experiments

Additional details for JRT-RNN To facilitate comparisons to prior work, we start with the Based architecture (Arora et al., 2024) and replace its linear attention layers with JRT-RNN linear attention layers. Note that the Based architecture hybridizes gated convolution layers (kernel size 3), sliding window attention layers (window size 128), and linear attention layers (using a Taylor approximation to the exponential function as the feature map, with feature dimension 16). We maintain the exact same order and number of each layer type as the Based work. We reduce the number of gated convolution layers by 1 at 360M parameters to account for the increase in parameters due to the encoder projections.

Next we include a description of the linear attention feature map used in our trained models. Based uses a 2nd-order Taylor approximation to the softmax-exponential function as the feature map $\phi : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ (Zhang et al., 2024). To approximate $\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d})$:

$$\exp(x) \approx 1 + x + \frac{x}{2!} \tag{6}$$

$$\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j) = 1 + \boldsymbol{q}_i^\top \boldsymbol{k}_j + \frac{(\boldsymbol{q}_i^\top \boldsymbol{k}_j)^2}{2} \tag{7}$$

The second order term has large dimension 273 if $\tilde{d} = 16$ as in (Arora et al., 2024). As a result, a careful IO-aware implementation is key to efficiency.

Training protocol For Table 2, we use the code provided by the baseline works, which has been adapted from the FlashAttention code base: `https://github.com/Dao-AILab/flash-attention/tree/main` for our pre-training runs (Dao, 2024). The Pile data is tokenized using the GPT2BPETokenizer and all models see the data in the same order. Here we provide details on the hyperaparamters and configurations used for training each architecture.

- **JRT-RNN** We provide hyperparameters and settings used for JRT-RNN in Table 15. We integrate JRT-RNN into the Based implementation released by the prior work.
- **Based (Arora et al., 2024)** We train using the specifications in Table 16 and the architecture implementation provided here: `https://github.com/HazyResearch/based`.
- **Transformer++ (Touvron et al., 2023)** We refer to the modern Llama architecture with Rotary encodings, RMSNorm and SwiGLU as Transformer++, following prior work (Gu and Dao, 2023; Yang et al., 2023). We train using the the specifications in Table 18 using the Flash Attention training code provided here: `https://github.com/Dao-AILab/flash-attention/tree/main` (Dao, 2024).
- **Mamba (Gu and Dao, 2023)** We train using the specifications in Table 17, where the parameters are sourced from the Appendix of (Gu and Dao, 2023). The architecture implementation is from the reference at `https://github.com/state-spaces/mamba`.

We give all models the Transformer++ change (e.g., SwiGLU, Rotary) where relevant.

Inference protocol For JRT-RNN, we left-pad prefill when it is shorter than the encoder region and mask in the linear attention layer following Listing 3 Appendix D. We apply no changes if the prefill exceeds the encoder region. For all results reported in this work, we use the parallel view of JRT-RNN to process the prefill and compute initial states following Section 4, then use the recurrent view to decode.

B.4. Additional details for Pile perplexity slicing analysis

In Appendix C.1, we analyze the perplexity of different models trained on the Pile, on the Pile test data. Here we provide additional details for the protocol.

We compute the training counts of bigrams across 10M Pile training documents, each of length 2048. We evaluate the models on $3,200$ sequences of length 2048 (6.6M total tokens), and measure perplexity on the last 1024 tokens per sequence (the causal, decoder region for JRT-RNN) (3.3M total tokens). We then evaluate perplexity on two slices of this test set:

1. *Associative recall (AR) hits.* Tokens in the final position of a bigram which previously occurred in context, and this bigram is infrequent during training. For instance, in the sequence "While lunching at the Maison Bergey bistro near his apartment: he had been musing about the ... (723 tokens) ... the young waitress's sigh at the Maison Bergey." the second "Bergey" would be included as an "AR hit" if "Maison Bergey" is a rare bigram during training. Intuitively, the model would need to rely on the context to predict the next token if the bigram were rare during training (i.e., was not memorized), testing the model's recall ability.
2. *Other tokens.* All other tokens. Intuitively, these tokens test the knowledge memorized in the model parameters.

In Figure 3, for the **recall frequencies** plot, we restrict to "AR hits" where the bigram and the re-occurrence of the bigram in context are separated by at least 1024 in distance within the context. In the **recall gaps** plot, we restrict to bigrams that are seen fewer than 1000 times during training and vary the distance between bigram occurrences in-context on the x axis.

B.5. Evaluation datasets

Here we provide additional details on the recall-intensive benchmark suite used in this work. The tasks include:

- **FDA** FDA is an information extraction task where documents are FDA reports for pre-market medical devices and the model needs to extract attributes such as the device code, classification, and indications for use (Arora et al., 2023b; 2024). These FDA reports are frequently analyzed by domain experts (Wu et al., 2021). We use the dataset released at: `https://huggingface.co/datasets/hazyresearch/based-fda`, which is part of the LM-Eval Harness repository (Gao et al., 2023).
- **SWDE** SWDE is an information extraction task where documents are HTML webpages spanning 14 different websites in the Movie and University topic domains (e.g., "IMDB.com", "RottenTomatoes", "USNews") and the model needs to extract attributes such as the Movie director / assistant director and University tuition (Lockard et al., 2020; Deng et al., 2022; Arora et al., 2023b; 2024). We use the dataset released at: `https://huggingface.co/datasets/hazyresearch/based-swde`, which is part of the LM-Eval Harness repository (Gao et al., 2023).
- **SQUADv2** SQUADv2 is a document QA benchhmark where documents come from Wikipedia and answer to questions are a span of tokens in the document (Rajpurkar et al., 2018; Arora et al., 2024). We use the version of the dataset released at: `https://huggingface.co/datasets/hazyresearch/based-squad`, which is part of the LM-Eval Harness repository (Gao et al., 2023).
- **TriviaQA** TriviaQA is a popular document QA benchmark where documents come from both Wikipedia and the general web and the question structure varies (Joshi et al., 2017). We use the dataset released at: `https://huggingface.co/datasets/mandarjoshi/trivia_qa`
- **Natural Questions (NQ)** Natural Questions is a popular document QA benchmark where documents come from Wikipedia and the questions are real queries issued to the Google search engine (Kwiatkowski et al., 2019). The answers are spans of text from the documents. We use the dataset released at: `https://huggingface.co/datasets/natural_questions`.
- **Drop** DROP is a challenging document QA benchmark that requires discrete reasoning over paragraphs from Wikipedia articles (Dua et al., 2019). The questions often require arithmetic operations, counting, or sorting of information found in the documents. We use the dataset released at: `https://huggingface.co/datasets/ucinlp/drop`.

Cloze Completion Formatting As the models in this work are not instruction fine-tuned and have been trained on next token prediction, they are more effective at producing relevant answers when the prompt format aligns with the pre-training task (next token prediction) as shown in prior work (Arora et al., 2022). Therefore, we reformat the questions in these benchmarks to a cloze-completion format using Meta's Llama-3-70B model (AI@Meta, 2024).

Given the question and the answer, the prompt we use is, where we provide the original question and answer from the task example:

Converting to Cloze Format

```
Can you rewrite this question and answer as a statement. Ensure that the answer is the last part of the
    statement.


Question: {question}

Answer: {answers}

Rewrite:
```

As an example:

Example

Input

```
Can you rewrite this question and answer as a statement. Ensure that the answer is the last part of the
    statement.


Question: Which team scored the final TD of the game?

Answer: Dallas

Rewrite:
```

Answer

```
The team that scored the final TD of the game is Dallas.
```

We filter the dataset by picking the rewrite with the answer appearing in the end and we remove the answer (e.g., "Dallas") when producing the final dataset. We report the resulting dataset sizes in Table 7 and release the datasets for reproducal.

Dataset	Size	Token
FDA	1102	1999.9
SWDE	1111	1036.1
SQUAD	2984	151.9
TriviaQA	1698	310.1
NQ	3157	8857.7
Drop	2084	236.6

Table 7: Evaluation Dataset Overview

Metrics We evaluate whether the model generated answer contains the exact answer span specified in the task. We run inference using the newline character and max generation length of 48 as stop-conditions.

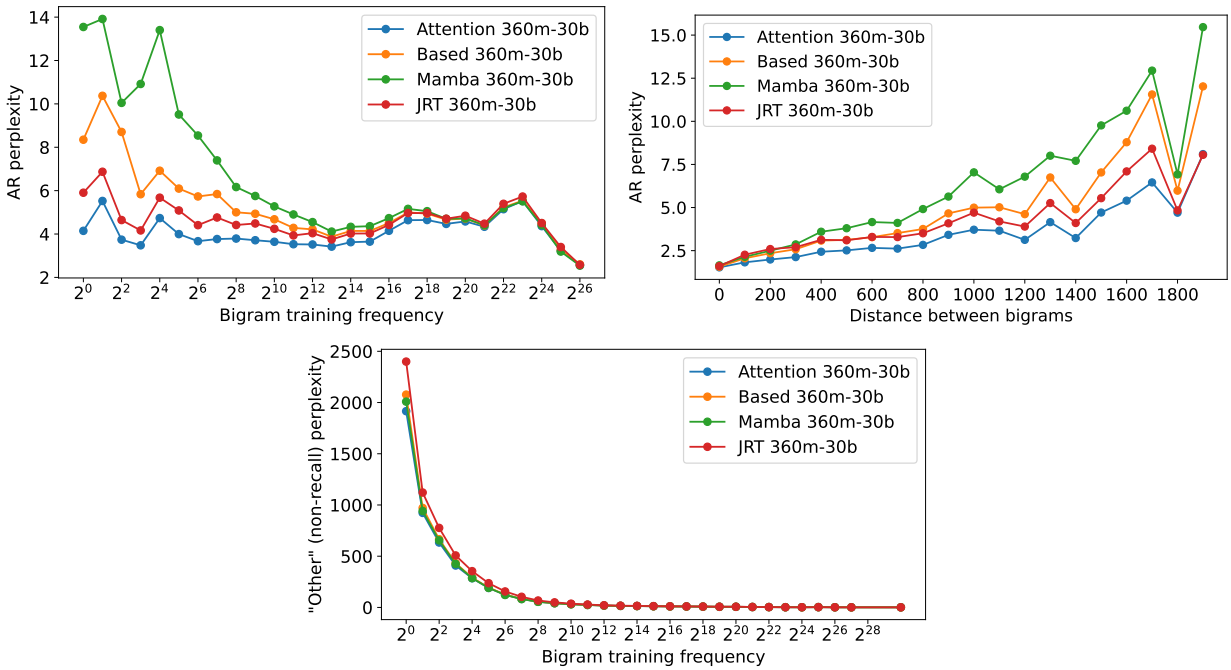

Figure 3: **Perplexity slices.** We slice the Pile test set perplexities of the pretrained LMs into associative recall "AR" and non-recall "Other" slices. A token is an AR token if it corresponds to a bigram that is re-occurring in the context, since the LM can look to the prior occurrence to predict the next token (Def. in Appendix C.1). **Top left (recall frequencies)** We plot y perplexity on AR bigram tokens that test the LMs' recall skills based on x the bigram frequency in training. **Top right (recall distances)** We plot y perplexity for AR tokens based on x the distances between the re-occuring bigrams in context. **Bottom (non-recall frequencies)** We plot y perplexity on non-recall tokens based on x the bigram frequency in training. Further details are in Appendix B.

C. Additional experiments

C.1. Overall language modeling

While we focus on a suite of recall-intensive benchmarks in Section 5, here we show that JRT-RNN maintains the quality of baseline models on other common in-context learning benchmarks.

Language modeling perplexity. A fundamental challenge is how to compare the inherent quality of models pre-trained with disparate objectives. In our setting, this is challenging since JRT-RNN additionally minimizes a masked language modeling objective beyond the standard causal next token prediction objective and sees 50% less data than the decoder-only models for the next token prediction task (when $M = 1024, N = 2048$). Overall JRT-RNN computes losses on 65% of the number of training data tokens seen by the decoder-only models (with 15% masked tokens in the encoder region).

Despite these differences, we consider a simple proxy of evaluating the perplexity of decoder-baselines in comparison to encoder-decoder JRT-RNN in the overlapping non-causal regions of both model types (i.e. the last 1024 tokens per input sequence of $N = 2048$ for our trained models). Following prior work (Arora et al., 2023a), we further *slice* the perplexity in two groups: (1) the associative recall "AR slice" includes tokens, referred to as "AR hits", that require the model to perform recall in order to predict the next token correctly and (2) the "Other slice" containing the remaining tokens (e.g., memorized knowledge). [6]

Slicing the model predictions on the Pile test set, we observe the following. Our measurement protocols are described in further detail in Appendix B.

1. **Recall frequencies.** JRT-RNN excels in the "AR slice". For infrequently seen bigrams during training (unlikely to be memorized in the model parameters), JRT-RNN improves in perplexity relative to Based and Mamba, two strong causal recurrent baselines (Figure 3, top right).

[6] As a heuristic rule, a token is an "AR hit" if it is completes a bigram that was previously seen in-context, and this bigram is infrequent during training (i.e., was not memorized by the model) (Arora et al., 2023a). For instance, in the sequence "In 1957, Dr. Seuss wrote ... In 1982, Dr. **Seuss**" the second **Seuss** would be included as an "AR hit" if "Dr. Seuss' is a rare bigram during training.

Model	Shots	BoolQ	CB		COPA	MultiRC	ReCoRD		RTE	WiC	WSC	Avg
		Acc. ↑	Acc. ↑	F1 ↑	Acc. ↑	Acc. ↑	F1 ↑	EM ↑	Acc. ↑	Acc. ↑	Acc. ↑	
JRT-RNN (356m/30b)	0	49.2	33.9	17.4	65.0	57.2	16.5	15.8	53.1	50.0	37.5	39.6
	1	46.5	37.5	26.9	65.0	51.9	18.9	18.1	46.2	46.6	55.8	41.3
	5	49.1	44.6	30.5	71.0	56.3	26.7	25.8	48.0	50.5	50.0	45.3
Based (360m/30b)	0	57.6	32.1	21.7	65.0	57.2	17.4	17.0	54.5	50.0	36.5	40.9
	1	54.9	35.7	25.7	70.0	55.3	21.8	21.1	48.0	48.1	55.8	43.6
	5	53.5	53.6	36.7	76.0	56.4	25.3	24.4	50.5	53.6	51.0	48.1
Transformer (360m/30b)	0	59.3	41.1	24.1	68.0	57.2	14.6	14.2	54.9	50.0	36.5	42.0
	1	54.9	37.5	26.9	70.0	54.2	21.1	20.4	43.7	46.4	53.8	42.9
	5	49.1	46.4	30.9	68.0	55.2	23.7	23.0	52.7	51.1	52.9	45.3
Mamba (358m/30b)	0	56.4	35.7	25.8	68.0	57.2	27.2	26.6	53.4	50.0	36.5	43.7
	1	51.1	41.1	28.5	70.0	52.3	25.8	25.1	50.2	46.4	55.8	44.6
	5	50.0	51.8	34.8	70.0	54.5	23.2	22.5	46.9	50.3	51.0	45.5

Table 8: **SuperGLUE benchmark evaluations.** We evaluate the models from Table 2 on the SuperGLUE benchmark (Wang et al., 2019) using the EleutherAI LM Eval harness (Gao et al., 2023).

Model	Shots	BoolQ	CB		COPA	MultiRC	RTE	WiC	WSC	Avg
		Acc. ↑	Acc. ↑	F1 ↑	Acc. ↑	Acc. ↑	Acc. ↑	Acc. ↑	Acc. ↑	
JRT-RNN	0	57.4	33.9	22.4	74.0	57.2	52.7	50.0	36.5	50.9
(1.3B/50B)	5	52.1	50.0	34.5	75.0	53.9	49.8	50.0	55.8	54.1
Based	0	55.1	41.1	19.4	71.0	56.8	53.1	50.0	53.8	52.9
(1.3B/50B)	5	52.5	50.0	33.7	75.0	51.4	49.1	53.1	53.8	53.8
Transformer	0	57.6	41.1	28.8	72.0	56.0	54.2	50.0	53.8	54.1
(1.3B/50B)	5	54.8	41.1	26.2	73.0	51.7	57.4	50.3	47.1	52.9
Mamba	0	54.8	25.0	25.2	73.0	56.4	51.3	50.0	40.4	50.1
(1.3B/50B)	5	55.6	53.6	45.5	75.0	53.7	53.8	51.7	56.7	56.6

Table 9: Same as Table 8 at the 1.3b parameter scale, trained on 50b tokens.

2. **Recall distances.** In the "AR slice", the gap between JRT-RNN and the decoder-only baselines grows as the distances between repeated bigrams seen in-context grows. This provides further support beyond Section 5.1 that JRT-RNN can help with longer context recall tasks (Figure 3).

3. **Non-recall frequencies.** JRT-RNN is worse in perplexity than the decoder-only LMs for the *non-recall* "Other slice" for bigrams that are rarely seen during training. This slice tests the model's use of memorized knowledge (as opposed to knowledge provided in the context). This is expected as JRT-RNN computes losses 65% of the tokens of the decoder-only LMs. We expect this gap to decrease with scale and longer training durations (seen as the bigram frequencies increases) (Figure 3, top left). Future work could also consider decoupling sequence mixers from MLPs (knowledge stores) in training. How best to normalize training between encoder-decoder and decoder-only LMs is an open question.

SuperGLUE Naural Language Understanding We use SuperGLUE (Wang et al., 2019) suite. We run these evaluations using the LM-Eval Harness repository's default settings (Gao et al., 2023).

In Table 8 and Table 9, we observe that all models achieve comparable quality. These results align with prior work suggesting that while alternate architectures provide similar overall language modeling perplexity, their quality on recall-intensive tasks is much more variable (Arora et al., 2023a; Gu and Dao, 2023; Akyürek et al., 2024; Arora et al., 2024).

Padding We note that the SuperGLUE inputs are quite short in sequence length, meaning that JRT-RNN sees pad tokens in the majority of the encoder region of the input until we reach length $M = 1024$. We use the space-token as the pad token in our evaluations, as discussed in Appendix B. Since we do not train with pad tokens in this work, this such sequences are relatively out of distribution, but with masking the padding portion of the sequence, we can recover quality. In Table 10, we evaluate JRT-RNN where we do not mask on the linear attention layers and observe quality starkly degrades on certain tasks (e.g., Copa and WSC).

Model	Shots	BoolQ	CB		COPA	MultiRC	ReCoRD		RTE	WiC	WSC	Avg
		Acc. ↑	Acc. ↑	F1 ↑	Acc. ↑	Acc. ↑	F1 ↑	EM ↑	Acc. ↑	Acc. ↑	Acc. ↑	
JRT-RNN	5	53.5	53.6	36.7	76.0	56.4	25.3	24.4	50.5	53.6	51.0	44.2
+No Pad Mask	5	49.1	55.4	38.2	56.0	56.3	26.7	25.8	51.6	49.7	40.4	41.3

Table 10: **Few-shot downstream evaluation on SuperGLUE of pre-trained language models.** Same protocol as Table 8, however **we do not mask the left-padding** in the linear attention layers.

C.2. JRT-RNN ablations

Training without MLM Loss JRT-RNN inspired by Prefix LM due to its simplicity. Prior work and our own finds that Prefix LM underperforms in quality (Wang et al., 2022). Here we compare JRT-RNN with and without the masked language modeling (MLM) loss. Excluding the MLM loss matches the protocol in prior Prefix-LM training. In Table 11, we find that the model is decent at longer sequences, but drops quality on short-context prompts.

	N=512		N=1024		N=2048	
	SWDE Acc. ↑	FDA Acc. ↑	SWDE Acc. ↑	FDA Acc. ↑	SWDE Acc. ↑	FDA Acc. ↑
Based	25.4	51.0	19.1	30.1	15.7	13.4
JRT-RNN, no MLM loss	23.9	38.7	21.6	39.2	18.5	18.3

Table 11: **Ablations of design choices in JRT-RNN** All models are 360M param variants of JRT-RNN, trained to 10 billion tokens on the Pile.

Training with Based ablations Based is a hybrid architecture with some linear attention, sliding window attention, and gated short-convolution layers. In Table 12, we train with the JRT-RNN vs. decoder-only approaches while ablating the mixture of layer types. The results suggest prefix linear attention remains useful for these recall-intensive tasks.

	N=512		N=1024		N=2048	
	SWDE Acc. ↑	FDA Acc. ↑	SWDE Acc. ↑	FDA Acc. ↑	SWDE Acc. ↑	FDA Acc. ↑
Linear attention (Taylor map)	29.6	25.5	21.5	16.0	23.0	4.6
Prefix linear attention (Taylor map)	36.8	57.7	27.1	48.7	23.9	8.2
Linear + Sliding attention	25.4	10.3	21.2	8.1	20.8	3.0
Prefix Linear + Sliding attention	35.5	53.3	34.8	46.5	32.1	30.0

Table 12: **Ablations of the types of sequence mixers in the LMs.** The default Based and JRT-RNN architectures in the main paper use a hybrid of sliding window attention (SWA), gated convolutions, and linear attention (LA). Here we also evaluate pure linear attention variations (top two rows, no SWA, no Convs.) and linear attention plus SWA (bottom two rows, no Convs.). All models are 360M param variants of JRT-RNN, trained to 30 billion tokens on the Pile using the same learning rates and schedules. In (Arora et al., 2024), it is also observed that the short convolution layers are helpful for such tasks.

D. JRT-RNN implementation details

In this section, we first provide a PyTorch reference for JRT-RNN and then discuss the IO-aware CUDA implementation.

D.1. Reference code for JRT-RNN

Below we include a PyTorch reference for the proposed layer, showing the parallel and recurrent views.

```python
from einops import rearrange
import torch
from torch import nn


def encoder(k, v):
    k, v = k.unsqueeze(-2), v.unsqueeze(-1)
    kv_state = (k * v).sum(dim=2, keepdim=True)
    k_state = k.sum(dim=2, keepdim=True)
    return kv_state, k_state

def decoder(q, k, v):
    q, k, v = q.unsqueeze(-2), k.unsqueeze(-2), v.unsqueeze(-1)
    kv_state_dec = (k * v).cumsum(dim=2)
    k_state_dec = k.cumsum(dim=2)
    return q, kv_state_dec, k_state_dec

def compute_linear_output(q_dec, k_dec, v_dec, k_enc, v_enc):
    kv_state_enc, k_state_enc = encoder(k_enc, v_enc)
    q, kv_state_dec, k_state_dec = decoder(q_dec, k_dec, v_dec)

    kv_state_dec = kv_state_enc + kv_state_dec
    k_state_dec = k_state_enc + k_state_dec

    z = 1 / ( q * k_state_dec).sum(dim=-1)
    y = ( (q * kv_state_dec).sum(dim=-1))
    output = y * z
    output = rearrange(output, 'b h l d -> b l (h d)')
    return output

def compute_parallel_output(q_dec, k_dec, v_dec, k_enc, v_enc):

    # Scaling
    k_state = k_enc.sum(dim=2, keepdim=True) +  k_dec.cumsum(2)
    z = 1 / ((q_dec * k_state).sum(dim=-1))

    # standard attention
    A_qk = torch.einsum("bhnd,bhmd->bhnm", q_dec, k_dec)
    A_qk = torch.tril(A_qk)
    y = torch.einsum("bhnm,bhme->bhne", A_qk.to(q_dec.dtype), v_dec.to(q_dec.dtype))
    y = y * z[..., None]
    output_1 = rearrange(y, 'b h l d -> b l (h d)')

    # cross attention
    A_qk_2 = torch.einsum("bhnd,bhmd->bhnm", q_dec, k_enc)
    y = torch.einsum("bhnm,bhme->bhne", A_qk_2.to(q_dec.dtype), v_enc.to(q_dec.dtype))
    y = y * z[..., None]
    output_2 = rearrange(y, 'b h l d -> b l (h d)')
    output_ref = output_1 + output_2
    return output_ref

# Inputs
enc_len, dec_len = seqlen // 2, seqlen
q_dec = torch.randn((batch, heads, dec_len, head_dim))
k_dec = torch.randn((batch, heads, dec_len, head_dim))
v_dec = torch.randn((batch, heads, dec_len, head_dim))
k_enc = torch.randn((batch, heads, enc_len, head_dim))
```

```
v_enc = torch.randn((batch, heads, enc_len, head_dim))

q_dec = feature_map(q_enc) # head_dim to expanded_dim
k_enc = feature_map(k_enc)
k_dec = feature_map(k_dec)

out = compute_linear_output(q_dec, k_dec, v_dec, k_enc, v_enc)
out_ref = compute_parallel_output(q_dec, k_dec, v_dec, k_enc, v_enc)
```

Listing 1: Minimal PyTorch implementation of JRT RNN.

```
if mask is not None and q.shape[2] > 1: # Check that we're in prefill
    if len(mask.shape) == 4:
        lin_attn_mask = (mask == 0)[:, :1, -1, :][..., None]  # b,1,k_len,1
    else:
        lin_attn_mask = mask[:, None, :, None]  # b,1,k_len,1
    lin_attn_mask = lin_attn_mask.to(torch.bool)
    k = k.masked_fill(~lin_attn_mask, 0)
    k_enc = k_enc.masked_fill(~lin_attn_mask, 0)
```

Listing 2: PyTorch implementation linear attention masking

D.2. IO-aware implementation

We build our implementation from the custom kernel for the Based architecture released in prior work (Arora et al., 2024) (Algorithm 1). [7] Letting $\mathrm{fn}_{\mathrm{based}}$ be the prior kernel, we use Algorithm 2 as the IO-aware implementation of JRT-RNN. We modify $\mathrm{fn}_{\mathrm{based}}$ to (1) avoid multiplications with queries in the first call and to simply compute the KV-state, and (2) we use the final row (row M) of the KV-state, representing the sum of $(\mathrm{k_e} * \mathrm{v_e})$ along the sequence dimension.

Algorithm 2 JRT-RNN CUDA Kernel Pseudocode

Require: Input decoder representations $q_d, k_d, v_d \in \mathbb{R}^{N \times d}$ and encoder representations $k_e, v_e \in \mathbb{R}^{M \times d}$.
Ensure: Output $y \in \mathbb{R}^{N \times d}$

Initialize SRAM buffers and register file fragments following Algorithm 1 (Arora et al., 2024). Including registers $A0, A1, A2$ to store the KV-state (for the $0^{th}, 1^{st}, 2^{nd}$ order terms of the Based linear attention kernel Taylor approximation respectively) and SRAM buffer y for storing the final output

Run $\mathrm{fn}_{\mathrm{based}}(\mathrm{k_e}, \mathrm{v_e})$ to compute KV-state for the encoder, where the result is held in registers $A0, A1, A2$. We modify the previously proposed Based implementation by using the non-causal sum instead of cumsum for the KV states. We don't multiply with queries in this step, as is done in the original algorithm.

Run $\mathrm{fn}_{\mathrm{based}}(\mathrm{q_d}, \mathrm{k_d}, \mathrm{v_d})$, from the register state initialized by the encoder computation. This computes the output y, held in SRAM.

Store y from SRAM to HBM.

[7] https://github.com/HazyResearch/ThunderKittens

E. Analysis

In this section, we provide qualitative analysis for JRT-PROMPT using three representative recurrent LMs, Mamba pretrained for 300b tokens on the Pile at the 370M, 1.4B, and 2.8B parameter scales.

We first bucket the common error modes, finding three primary categories: (1) No Answer (N/A), (2) Repetition, and (3) Irrelevant outputs. The statistics for each category are shown in Table 13. Compared to the standard default zero-shot prompting approach, JRT-PROMPT tends to increase the No Answer error and repetition errors, while reducing errors related to irrelevant outputs.

Model	Mamba-370m			Mamba-1.4B			Mamba-2.8B		
Error Type	N/A	Rep	Irrel	N/A	Rep	Irrel	N/A	Rep	Irrel
FDA-default	0.2	35.4	22.7	0.1	31.1	23.0	0.2	27.5	18.3
FDA-JRT-PROMPT	0.0	29.4	**12.3**	0.1	29.2	**9.8**	0.0	23.3	**9.8**
SWDE-default	39.1	20.2	13.1	37.3	17.3	7.8	32.3	18.9	9.7
SWDE-JRT-PROMPT	23.6	17.0	17.2	28.0	15.0	11.1	26.9	14.7	9.6
SQUAD-default	0.0	6.6	58.6	0.0	5.9	54.2	0.0	5.5	51.3
SQUAD-JRT-PROMPT	0.0	12.2	**37.0**	0.1	10.7	**30.0**	1.6	32.9	**13.8**

Table 13: **Error Mode Statistics** We calculate the percentage ratio of different error types to the total number of test data points. N/A: No Answer; Rep: Repetition; Irrel: Irrelevant.

No Answer One error observed in the models is the output of an empty string, especially in tasks with complex text. We believe this is due to formatting sensitivity and could reduce with model scale.

No Answer Example

Input

```
Information about the applicant in the text: SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A. 510(
     k) Number: K172333 B. Purpose for Submission: To expand the use of previously cleared assay reagents
     for Factor V Leiden; ...... D. Type of Test: Quantitative clot-based applications E. Applicant: Siemens
      Healthcare Diagnostics Product GmbH F. Proprietary and Established Names: ...... G. Regulatory
     Information: ...... Protein C with Protein C Reagent Antithrombin (AT) with INNOVANCE Antithrombin
     Protein C with Berichrom Protein C \n
Information about the applicant in the text: SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A. 510(
     k) Number: K172333 B. Purpose for Submission: To expand the use of previously cleared assay reagents
     for Factor V Leiden; ...... D. Type of Test: Quantitative clot-based applications E. Applicant: Siemens
      Healthcare Diagnostics Product GmbH F. Proprietary and Established Names: ...... G. Regulatory
     Information: ...... Protein C with Protein C Reagent Antithrombin (AT) with INNOVANCE Antithrombin
     Protein C with Berichrom Protein C applicant:
```

Prediction

```
""
```

Ground Truth

```
Siemens Healthcare Diagnostics Product GmbH
```

Repetition If the model reads repeated phrases (e.g., documents and questions), it may merely repeat the document and question again rather than providing an answer, when using JRT-PROMPT. These models are not instruction fine-tuned and identifying the relevant task may be difficult.

Repetition Error Example

Input

```
Information about the applicant in the text: SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A. 510(
     k) Number: K182513 B. Purpose for Submission: New device 510(k) clearance for the FluChip-8G Influenza
     A+B Assay C. Measurand: Influenza A and influenza B viral nucleic acids. ...... E. Applicant: InDevR,
     Inc. F. ...

Information about the applicant in the text:) SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A.
     510(k) Number: K182513 B. Purpose for Submission: New device 510(k) clearance for the FluChip-8G
     Influenza A+B Assay C. Measurand: Influenza A and influenza B viral nucleic acids. ...... E. Applicant:
      InDevR, Inc. F. ...
applicant:
```

Prediction

```
Information about the applicant in the text:) SUBSTANTIAL EQUIVALENCE DETERMINATION SUMMARY A. 510(k)
     Number: K182513 B. Purpose for Submission: New device 510(k) clearance
```

Ground Truth

```
InDevR, Inc.
```

Irrelevant Output Sometimes model outputs are undesirable and unrelated to the input text. For instance, the model may provide new *continuations* of the text as opposed to referring back to the context and outputting previously seen information. JRT-PROMPT appears to help reduce these types of errors.

Irrelevant Output Example

Input

```
 "Title: Martin_Luther\nBackground: At the heart of scholars' debate about Luther's influence is whether it
      is anachronistic to view his work as a precursor of the racial antisemitism of the Nazis...
 Title: Martin_Luther\nBackground: At the heart of scholars' debate about Luther's influence is whether it
     is anachronistic to view his work as ...... His position was entirely religious and in no respect
     racial.\"Martin Brecht referred to Luther's stand on the Jews as
```

Prediction

```
a very important and important part of the history of the German people.
```

Ground Truth

```
misguided agitation
```

Few shot prompting A common hypothesis for why few-shot prompting is more effective than zero-shot prompting is that it provides the model with a better understanding of the task at hand. Here we evaluate the few-shot baselines on recall-intensive tasks.

The in-context learning results for different models are shown in Table 14. The improvement of few-shot in-context learning in smaller models is less obvious than in larger models. JRT-PROMPT appears more effective than few-shot ICL on average, suggesting that there is benefit from reading twice, beyond simply improving the model's understanding of the task via few-shot examples.

One failure mode we observe with few-shot prompts is that the model sometimes outputs the attribute-value (e.g. director name given HTML text from different movie web pages) from the example documents instead of the relevant input document from which we seek to extract information.

	Mamba-130m			Mamba-370m			Mamba-1.4B			Mamba-2.8B		
	DF	FS	JP	DF	FS	JP	DF	FS	JP	DF	FS	JP
FDA	25.7	22.0	32.8	41.9	35.3	58.3	45.8	46.0	60.9	54.3	54.8	66.6
SWDE	17.5	19.7	31.5	27.6	35.0	42.2	37.6	47.1	46.0	38.9	51.9	48.9
SQUAD	27.1	25.2	51.9	34.9	36.0	51.0	39.9	45.5	59.6	43.9	53.2	59.4

Table 14: **JRT-PROMPT ablations**. Here we evaluate three ICL baselines: DF is default prompt; FS is a prompt with 2 in-context examples; JP is JRT-PROMPT.

F. Prompts

Below we include the prompts for the default and JRT-PROMPT in-context learning results that produced the numbers in Table 1. We use the exact same prompt structure for all examples in the task and across all models. We use a shared structure across groups of tasks e.g., information extraction tasks SWDE and FDA use the same prompt structure and document QA tasks (NQ, TriviaQA, Drop, SQUAD).

F.1. SWDE

SWDE (Default)

Input

```
The Evil Dead Movie Facts and Details click here amc home | movie guide Genres\nLists\nRatings amctv.com>
     movie guide>The Evil Dead>details The Evil Dead details\nOverall Rating Total Ratings: 1 Overview\
     nDetails\nCast & Credits\nAwards\nReview Movie Details: Director: Sam Raimi\nProduced By: New Line
     Cinema, Renaissance Pictures\nYear: 1983\nRun Time: 85 minutes\nCountry: USA\nLanguage: English MPAA
     Rating: R\nCategory: Feature\nGenre/Type: Horror\nFilmed In: Color Key Cast: Bruce Campbell, Ellen
     Sandweiss, Betsy Baker, Hal Delrich

... many document tokens ...

cranked up the story's comic aspects several dozen notches for the rollicking semi-remake, Evil Dead 2:
     Dead by Dawn. by Cavett Binion, Rovi Keywords: atrocity\nbook\ncabin\ncellar\nchainsaw\ndemon\
     ndismemberment\ngateway-to-hell\nmonster\ndemonic-possession rampage\nsatanic\nSatanism\nslasher\ntree\
     nweekend\nwoods [place]\ncollege-student\ninvocation Themes: Zombies\nDemonic Possession\nNightmare
     Vacations\nCurses and Spells Exclusive coverage Get Dragged to Hell With This Ultimate Sam Raimi Fan
     Quiz - Horror Hacker - AMCfrom AMC Blogs\nInside the Unlikely Cult of Road House - AMC Movie Blog -
     AMCfrom AMC Blogs\nU.S. Marshals and Five Other Stealth. Year:
```

Ground Truth

```
1983
```

SWDE (Twice)

Input

```
Information about Year. The Evil Dead Movie Facts and Details click here amc home | movie guide Genres\
     nLists\nRatings amctv.com>movie guide>The Evil Dead>details The Evil Dead details\nOverall Rating Total
      Ratings: 1 Overview\nDetails\nCast & Credits\nAwards\nReview Movie Details: Director: Sam Raimi\
     nProduced By: New Line Cinema,

... many document tokens ...

U.S. Marshals and Five Other Stealth.
The Evil Dead Movie Facts and Details click here amc home | movie guide Genres\nLists\nRatings amctv.com>
     movie guide>The Evil Dead>details The Evil Dead details\nOverall Rating Total Ratings: 1 Overview\
     nDetails\nCast & Credits\nAwards\nReview Movie Details: Director: Sam Raimi\nProduced By: New Line
     Cinema, Renaissance Pictures\nYear: 1983

... many document tokens ...

 With This Ultimate Sam Raimi Fan Quiz - Horror Hacker - AMCfrom AMC Blogs\nInside the Unlikely Cult of
     Road House - AMC Movie Blog. Year:
```

Ground Truth

```
1983
```

F.2. Natural Questions

Natural Questions (Default)

Input

```
List of Nobel laureates in Physics - wikipedia <H1> List of Nobel laureates in Physics </H1> Jump to :
     navigation, search Front side ( obverse ) of the Nobel Prize Medal for Physics presented to Edward
     Victor Appleton in 1947 <P> The Nobel Prize in Physics ( Swedish : Nobelpriset i fysik ) is awarded
     annually by the Royal Swedish Academy of Sciences to scientists in the various fields of physics.

... many document tokens ...


The first Nobel Prize in Physics was awarded to
```

```
Wilhelm Conrad Rontgen, of Germany
```

Natural Questions (Twice)

Input

```
Who got the first nobel prize in physics? List of Nobel laureates in Physics - wikipedia <H1> List of Nobel
      laureates in Physics </H1> Jump to : navigation, search Front side ( obverse ) of the Nobel Prize
     Medal for Physics presented to Edward Victor Appleton in 1947 <P> The Nobel Prize in Physics ( Swedish
     : Nobelpriset i fysik ) is awarded annually by the Royal Swedish Academy of Sciences to scientists in
     the various fields of physics.

... many document tokens ...

for their joint researches on the radiation phenomena discovered by Professor Henri Becquerel
List of Nobel laureates in Physics - wikipedia <H1> List of Nobel laureates in Physics </H1> Jump to :
     navigation, search Front side ( obverse ) of the Nobel Prize Medal for Physics presented to Edward
     Victor Appleton in 1947 <P> The Nobel Prize in Physics ( Swedish : Nobelpriset i fysik ) is awarded
     annually by the Royal Swedish Academy of Sciences to scientists in the various fields of physics.

... many document tokens ...

for their joint researches on the radiation phenomena discovered by Professor Henri Becquerel. The first
     Nobel Prize in Physics was awarded to
```

```
Wilhelm Conrad Rontgen, of Germany
```

F.3. FDA

FDA (Default)

Input

```
510(k) SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A. 510(k) Number: K153137 B. Purpose for
     Submission: Clearance of a new device C. Measurand: Anti-PF4/Heparin Total Antibodies D. Type of Test:
     Automated, latex enhanced immuno-turbidimetric assay E. Applicant: Instrumentation Laboratory (IL) Co.
     F. Proprietary and Established Names:
HemosIL HIT-Ab
HemosIL HIT-Ab
Controls G. Regulatory Information: 1. Regulation section: 21 CFR 864.7695, Platelet factor 4
     radioimmunoassay 21 CFR 864.5425, Multipurpose system for in vitro coagulation studies 2.

... many document tokens ...

Low HIT Control:
Control intended for the assessment of precision and accuracy of the assay at PF4/H antibody levels at or
     below the cut-off.
High HIT Control: Control intended for the assessment of precision and accuracy of the assay at abnormal
     PF4/H antibody levels. J. Substantial Equivalence Information: 1.
Predicate device name(s): Asserachrom HPIA Test kit from Diagnostica Stago 2. Predicate 510(k) number(s):
     K003767 3. Comparison with predicate: 4 Similarities Item Device Predicate Trade Names HemosIL HIT-Ab(
     PF4-H) HemosIL HIT-Ab (PF4-H) Controls (K153137) Asserachrom HPIA Test Kit (kit includes two control
     levels) (K003767) Measurand Anti-PF4/Heparin Total Antibodies AntiPF. Purpose for submission:
```

```
Clearance of a new device
```

FDA (Twice)

Input

```
Information about Purpose for submission. 510(k) SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A.
     510(k) Number: K153137 B. Purpose for Submission: Clearance of a new device C. Measurand: Anti-PF4/
     Heparin Total Antibodies D. Type of Test: Automated, latex enhanced immuno-turbidimetric assay E.
     Applicant: Instrumentation Laboratory (IL) Co. F.

... many document tokens ...

Predicate device name(s): Asserachrom HPIA Test kit from Diagnostica Stago 2. Predicate 510(k) number(s):
     K003767 3. Comparison with predicate: 4 Similarities Item Device Predicate Trade Names HemosIL HIT-Ab(
     PF4-H) HemosIL HIT-Ab(PF4-H) Controls (K153137) Asserachrom HPIA Test Kit (kit includes two control
     levels) (K003767) Measurand Anti-PF4/Heparin Total Antibodies Anti-PF.
510(k) SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY A. 510(k) Number: K153137 B. Purpose for
     Submission: Clearance of a new device C. Measurand: Anti-PF4/Heparin Total Antibodies D. Type of Test:
     Automated, latex enhanced immuno-turbidimetric assay E. Applicant: Instrumentation Laboratory (IL) Co.
     F.

... many document tokens ...

Predicate device name(s): Asserachrom HPIA Test kit from Diagnostica Stago 2. Predicate 510(k) number(s):
     K003767 3. Comparison with predicate: 4 Similarities Item Device Predicate Trade Names
HemosIL HIT-Ab(PF4-H) HemosIL HIT-Ab(PF4-H) Controls (K153137) Asserachrom HPIA Test Kit (kit includes two
     control levels) (K003767)
Measurand Anti-PF4/Heparin Total Antibodies Anti-PF. Purpose for submission:
```

```
Clearance of a new device
```

F.4. SQUAD

SQUAD (Default)

Input

```
Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL)
      for the 2015 season.
The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (
     NFC) champion Carolina Panthers 24-10 to earn their third Super Bowl title.
The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara,
     California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with
     various gold-themed initiatives, as well as temporarily suspending the tradition of naming each
Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that
      the logo could prominently feature the Arabic numerals 50.The NFL team that represented the AFC at
     Super Bowl 50 was the
```

```
Denver Broncos
```

SQUAD (Twice)

Input

```
Which NFL team represented the AFC at Super Bowl 50? Super Bowl 50 was an American football game to
     determine the champion of the National Football League (NFL) for the 2015 season. The American Football
      Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion
     Carolina Panthers 24-10 to earn their third Super Bowl title. The game was played on February 7, 2016,
     at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super
     Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as
     temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which
     the game would have been known as "Super Bowl L"), so that the logo could prominently feature the
     Arabic numerals 50.
Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL)
      for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the
     National Football Conference (NFC) champion Carolina Panthers 24-10 to earn their third Super Bowl
     title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at
     Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary
     " with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each
     Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so
     that the logo could prominently feature the Arabic numerals 50.The NFL team that represented the AFC at
      Super Bowl 50 was the
```

```
Denver Broncos
```

F.5. TriviaQA

TriviaQA (Default)

Input

```
81 years since the first inflight movie was shown...81 years since the first inflight movie was shown -
     Travelers United Travelers United 81 years since the first inflight movie was shown
October 8, 2010 Filed Under: Today By Charlie Leocha Leave a Comment Our government at work - This is the
     daily ''Profile America'' feature from the U.S. Census Bureau for today, Friday, October 8th.
This is the 81st anniversary of the first inflight movie ever shown. A little-known travel gem.
Friday, October 8th, celebrates one of the few joys left in long-distance flying, sitting back and enjoying
      a feature-length movie.
But recently, one major airline announced it will be ending this entertainment, joining several low-cost
     airlines in the policy.
While movies have been generally available on long flights for decades, the first movies shown in the air
     were a newsreel and two cartoons.
These were shown on this date in 1929 aboard a Ford Trimotor operated by Transcontinental Air Transport.
     Regular in-flight movie service began in July 1961 on a Trans World airline flight from New York to Los
      Angeles.
Now, more than 3.9 million passengers fly between New York and Los Angeles every year. You can find these
     and more facts about America from the U.S. Census Bureau online. The first in-flight movie was shown on
      an internal flight in the USA in
```

```
1929
```

TriviaQA (Twice)

Input

```
In what year was the first in-flight movie shown on an internal flight in the USA? 81 years since the first
      inflight movie was shown...81 years since the first inflight movie was shown - Travelers United
     Travelers United  81 years since the first inflight movie was shown  October 8, 2010  Filed Under:
     Today By Charlie Leocha Leave a Comment  .... These were shown on this date in 1929 aboard a Ford
     Trimotor operated by Transcontinental Air Transport. Regular in-flight movie service began in July 1961
      on a Trans World airline flight from New York to Los Angeles. Now, more than 3.9 million passengers
     fly between New York and Los Angeles every year. You can find these and more facts about America from
     the U.S. Census Bureau online at.
81 years since the first inflight movie was shown...81 years since the first inflight movie was shown -
     Travelers United  Travelers United  81 years since the first inflight movie was shown  October 8, 2010
      Filed Under: Today By Charlie Leocha Leave a Comment  ...  These were shown on this date in 1929
     aboard a Ford Trimotor operated by Transcontinental Air Transport. Regular in-flight movie service
     began in July 1961 on a Trans World airline flight from New York to Los Angeles. Now, more than 3.9
     million passengers fly between New York and Los Angeles every year. You can find these and more facts
     about America from the U.S. Census Bureau online at. The first in-flight movie was shown on an internal
      flight in the USA in
```

```
1929
```

F.6. Drop

Drop (Default)

Input

```
Hoping to rebound from their loss to the Patriots, the Raiders stayed at home for a Week 16 duel with the
     Houston Texans.  Oakland would get the early lead in the first quarter as quarterback JaMarcus Russell
     completed a 20-yard touchdown pass to rookie wide receiver Chaz Schilens.  The Texans would respond
     with fullback Vonta Leach getting a 1-yard touchdown run, yet the Raiders would answer with kicker
     Sebastian Janikowski getting a 33-yard and a 30-yard field goal.  Houston would tie the game in the
     second quarter with kicker Kris Brown getting a 53-yard and a 24-yard field goal. Oakland would take
     the lead in the third quarter with wide receiver Johnnie Lee Higgins catching a 29-yard touchdown pass
     from Russell, followed up by an 80-yard punt return for a touchdown.  The Texans tried to rally in the
     fourth quarter as Brown nailed a 40-yard field goal, yet the Raiders' defense would shut down any
     possible attempt. The first touchdown of the game was scored by
```

```
Chaz Schilens
```

Drop (Twice)

Input

```
Who scored the first touchdown of the game? Hoping to rebound from their loss to the Patriots, the Raiders
     stayed at home for a Week 16 duel with the Houston Texans.  Oakland would get the early lead in the
     first quarter as quarterback JaMarcus Russell completed a 20-yard touchdown pass to rookie wide
     receiver Chaz Schilens.  The Texans would respond with fullback Vonta Leach getting a 1-yard touchdown
     run, yet the Raiders would answer with kicker Sebastian Janikowski getting a 33-yard and a 30-yard
     field goal.  Houston would tie the game in the second quarter with kicker Kris Brown getting a 53-yard
     and a 24-yard field goal. Oakland would take the lead in the third quarter with wide receiver Johnnie
     Lee Higgins catching a 29-yard touchdown pass from Russell, followed up by an 80-yard punt return for a
      touchdown.  The Texans tried to rally in the fourth quarter as Brown nailed a 40-yard field goal, yet
     the Raiders' defense would shut down any possible attempt.
Hoping to rebound from their loss to the Patriots, the Raiders stayed at home for a Week 16 duel with the
     Houston Texans.  Oakland would get the early lead in the first quarter as quarterback JaMarcus Russell
     completed a 20-yard touchdown pass to rookie wide receiver Chaz Schilens.  The Texans would respond
     with fullback Vonta Leach getting a 1-yard touchdown run, yet the Raiders would answer with kicker
     Sebastian Janikowski getting a 33-yard and a 30-yard field goal.  Houston would tie the game in the
     second quarter with kicker Kris Brown getting a 53-yard and a 24-yard field goal. Oakland would take
     the lead in the third quarter with wide receiver Johnnie Lee Higgins catching a 29-yard touchdown pass
     from Russell, followed up by an 80-yard punt return for a touchdown.  The Texans tried to rally in the
     fourth quarter as Brown nailed a 40-yard field goal, yet the Raiders' defense would shut down any
     possible attempt. The first touchdown of the game was scored by
```

```
Chaz Schilens
```

G. Theoretical results

We begin by setting notation.

Notation. We will be denoting the all 1 row vector of size k, given by $\begin{bmatrix}1 & 1 & \dots & 1 & 1\end{bmatrix}$, and the all 0 row vector of size k, given by $\begin{bmatrix}0 & 0 & \dots & 0 & 0\end{bmatrix}$, as $\mathbf{1}^k$ and $\mathbf{0}^k$, respectively. We will also construe the standard basis vector $\mathbf{e}_i$ as a column vector in these notes, and adhere to the following matrix indexing convention: $\mathbf{M}[i,j]$ is the entry in the ith row and the jth column, $\mathbf{M}[i,:] \in \mathbb{F}^{1\times n}$ denotes the ith row, and $\mathbf{M}[:,j] \in \mathbb{F}^{m\times 1}$ denotes the jth column of $\mathbf{M} \in \mathbb{F}^{m\times n}$, where $\mathbb{F}$ is a field and the reader can substitute $\mathbb{F}$ for $\mathbb{R}$ for convenience. We then use $\mathbf{1}^{m\times n}, \mathbf{0}^{m\times n} \in \mathbb{F}^{m\times 1}$ to denote the matrix of all 1s and 0s, respectively.

Next, we denote the *Hadamard product* of vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$ as $\mathbf{u} \odot \mathbf{v}$; the operation can be extended to matrices by applying the Hadamard product column-wise across the matrices. This is commonly referred to as *(element-wise) gating*. For vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$, we also denote their *linear (or acyclic) convolution* as $\mathbf{u} * \mathbf{v}$ and *cyclic convolution* as $\mathbf{u} \circledast \mathbf{v}$.

We also recall the definition of `BaseConv` for the reader's convenience:

Definition G.1 (`BaseConv` (Arora et al., 2023a))**.** *Given an input sequence $\boldsymbol{u} \in \mathbb{R}^{N\times d}$, where N is the sequence length and d is the model dimension, a learned weight matrix $\boldsymbol{W}^B \in \mathbb{R}^{d\times d}$ and biases $\boldsymbol{B}^B, \boldsymbol{B}^K \in \mathbb{R}^{N\times d}$ and a matrix of convolution filters $\boldsymbol{K} \in \mathbb{R}^{N\times d}$, a `BaseConv` layer computes the following:*

$$\boldsymbol{z}^{\texttt{BaseConv}} := (\boldsymbol{u}\boldsymbol{W}^B + \boldsymbol{B}^B) \odot \left(\boldsymbol{K} * \boldsymbol{u} + \boldsymbol{B}^K\right) \in \mathbb{R}^{N\times d}, \tag{8}$$

where the convolutions are applied across the input length N.

We will need the following "5-tuple" notation for `BaseConv` model:

Definition G.2. *An $\left(N, L, d, \tilde{N}, \tilde{d}\right) -$ -`BaseConv` is a stacked sequence to sequence model with L layers such that:*

1. *input and output are $N \times d$ matrices,*
2. *each layer corresponds to the a `BaseConv` layer as defined in Definition G.1, and*
3. *all the individual gated convolution layers take in $\tilde{N} \times \tilde{d}$ matrices and output $\tilde{N} \times \tilde{d}$ matrices. We refer to the tuple $(\tilde{N}, \tilde{d})$ as the* inner dimension *of the model.*

We also assume that the input $\boldsymbol{u} \in \mathbb{R}^{N\times d}$ is embedded into $\boldsymbol{u}' \in \mathbb{R}^{\tilde{N}\times\tilde{d}}$ such that

$$\boldsymbol{u}'[n,t] = \begin{cases} \boldsymbol{u}[n,t] & \text{if } n < N,\ t < d \\ 0 & \text{otherwise.} \end{cases}$$

The output from the last layer $\boldsymbol{z} \in \mathbb{R}^{\tilde{N}\times\tilde{d}}$ is transformed into output $\boldsymbol{y} \in R^{N\times d}$ by extracting the top left $N \times d$ entries in $\boldsymbol{z}$.

Definition G.3. *An MLP layer is map $\mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ defined via matrices $\boldsymbol{W}^1, \boldsymbol{W}^2 \in \mathbb{R}^{d\times d}$ and "bias" matrices $\boldsymbol{B}^1, \boldsymbol{B}^2 \in \mathbb{R}^{N\times d}$ as follows:*

$$\texttt{MLP}(\boldsymbol{u}) = \mathrm{ReLU}(\boldsymbol{u}\boldsymbol{W}^1 + \boldsymbol{B}^1)\boldsymbol{W}^2 + \boldsymbol{B}^2.$$

G.1. JRT Lower Bounds for `BaseConv`

First, we formally define JRT prompts below.

Definition G.4 (JRT Prompts)**.** *For any model $\mathcal{M}$ with input $\boldsymbol{u} \in \mathbb{R}^{N\times d}$, a* JRT prompt *for input $\boldsymbol{u}$ is the repeated input $\boldsymbol{u}^{\mathrm{JRT}} \in \mathbb{R}^{2N\times d}$ given by*

$$\boldsymbol{u}^{\mathrm{JRT}}[i,:] := \begin{cases} \boldsymbol{u}[i,:] & \textit{if } i < N \\ \boldsymbol{u}[i-N,:] & \textit{otherwise.} \end{cases}$$

G.1.1. Lower Bound on the Number of Layers for AR

In this section, we will provide a lower bound on the number of layers needed to solve the standard associative recall problem with JRT prompts. We formally recall the associative recall problem:

> The AR problem takes key-value pairs $\{\boldsymbol{k}_i, \boldsymbol{v}_i\}_{i=0}^{N-1}$ along with a query $\boldsymbol{q}$ appended at the end as input and the goal is to output $\boldsymbol{v}_i$ if $\boldsymbol{q} = \boldsymbol{k}_i$ for some $i \in [0, N-1]$.

We also require a randomized communication complexity lower bound result for the *index problem*:

> The index problem has two agents, Alice and Bob, where Alice has a string $\boldsymbol{x} \in \{0,1\}^n$ and Bob has an index $i \in [n]$, and the goal for the players is to output the i-th entry $\boldsymbol{x}_i$. Moreover, we also require the communication to be *one-way*: only Alice is allowed to send a single message to Bob and Bob needs to output the answer.

We will use the following well-known lower bound for the index problem.

Theorem G.5 ((Jayram et al., 2008))**.** *The one-way randomized communication complexity*[8] *of the index problem for an n-length bit string is $\Omega(n)$.*

We will now mirror the argument from (Arora et al., 2024, Theorem F.4) to show that the lower bound on the number of layers for a `BaseConv` model solving AR still holds for JRT prompts.

Theorem G.6. *Given a* JRT prompt $\boldsymbol{u}^{\mathrm{JRT}} \in \{0,1\}^{2N \times d}$ *for input* $\boldsymbol{u} \in \{0,1\}^{N \times d}$ *to the AR problem with any encoding such that* $\log c \leq d \leq 2^{(\log N)^{1-\epsilon}}$ *for* $\epsilon > 0$*, and c possible tokens from the vocabulary with $c \leq N$, a data-independent* `BaseConv` *model with model parameters taking $O(\log N)$ bits needs $\Omega(\epsilon \log \log N)$ layers to solve AR.*

Proof. Given a `BaseConv` model $\mathcal{M}$ solving AR, regardless of the input length N, we know that there exists an equivalent polynomial $P(\boldsymbol{u}^{\mathrm{JRT}})$ of degree at most 2^L that solves AR for any $\boldsymbol{u}^{\mathrm{JRT}} \in \{0,1\}^{2N \times d}$, where L denotes the number of layers.[9] Now, take the instance $(\boldsymbol{x}, i)$ of the index problem with $\boldsymbol{x} \in \{0,1\}^N$ and the corresponding JRT prompt of the AR problem as before

$$\boldsymbol{u}^{\mathrm{JRT}} := \{j, \boldsymbol{x}_j\}_{j=0}^{N-1}, i, \{j, \boldsymbol{x}_j\}_{j=0}^{N-1}, i \tag{9}$$

Next, we build the following one-way protocol for solving the index problem using the `BaseConv` model from the hypothesis that it solves AR. Alice with their access of $\boldsymbol{x} \in \{0,1\}^N$ will again generate a JRT input $\boldsymbol{u}^{\mathrm{JRT}}$ for AR (without the query) as in equation 9. More specifically, Alice takes the values $\boldsymbol{a} := \boldsymbol{u}^{\mathrm{JRT}}[0 : N-2, :] \equiv \boldsymbol{u}^{\mathrm{JRT}}[N : 2N-2, :] \in \{0,1\}^{2(N-1) \times d}$ while leaving out the query $\boldsymbol{q} := \boldsymbol{u}^{\mathrm{JRT}}[N-1, :] = \boldsymbol{u}^{\mathrm{JRT}}[2N-1, :]$, and substitutes these known $2(N-1)d$ values to define the following polynomial:

$$Q^{\mathrm{JRT}}(\boldsymbol{q}) = P(\boldsymbol{a}, \boldsymbol{q}, \boldsymbol{a}, \boldsymbol{q}). \tag{10}$$

Crucially, Q^{JRT} is still a polynomial in d variables, corresponding to the values $\boldsymbol{u}^{\mathrm{JRT}}[N-1, :] = \boldsymbol{u}^{\mathrm{JRT}}[2N-1, :]$ that Bob has and trivially has degree $D \leq 2^L$. As in the proof of (Arora et al., 2024, Theorem F.4), Alice can run the model $\mathcal{M}$, retrieve the coefficients of Q^{JRT}, and send it to Bob. Since we assume that P solves AR, Bob can take the coefficients of Q^{JRT} and substitute $\boldsymbol{u}^{\mathrm{JRT}}[N-1, :] = \boldsymbol{u}^{\mathrm{JRT}}[2N-1, :]$ to Q^{JRT} to compute $P(\boldsymbol{u}^{\mathrm{JRT}})$ which is the value $\boldsymbol{x}_i$.

Moreover, the polynomial Q^{JRT} that Alice sends still has at most d^{2^L} coefficients as each term in Q^{JRT} can have degree at most 2^L. If each such coefficient has B bits, then using theorem G.5, the total number of bits being communicated must satisfy $B \cdot d^{2^L} \geq \Omega(N)$. This follows from the fact that if $B \cdot d^{2^L} \leq o(N)$, then since the associated value of i in equation 9 is the answer to the indexing problem, we have shown that a one-way communication protocol for solving the index problem uses $o(N)$ communication complexity, which then contradicts theorem G.5. This is the same equation we get in the proof of (Arora et al., 2024, Theorem F.4), which yields the following lower bound on the number of layers:

$$L \geq \log\left(\frac{\log N - \log B}{(\log N)^{1-\epsilon}}\right). \tag{11}$$

[8] The randomized communication complexity of function f is defined as $\min_\pi \|\pi\|$, where π ranges over all randomized protocols that can solve f with probability of success at least $2/3$.

[9] See the proof of (Arora et al., 2024, Theorem F.4) for justification.

Recall here that the model parameters are assumed to be $O(\log N)$ bits, so any coefficient in Q^{JRT} should have absolute value at most $\left(2^{O(\log N)} \cdot 2Nd\right)^{2^L}$ as each coefficient can be a product of at most $2Nd$ variables. That is, for some $\alpha > 0$, we have the following bound on each coefficient:

$$2^B \leq (2 \cdot N^{\alpha+1} d)^{2^L} \leq (2N^{(\alpha+2)})^{2^L}$$

where the last equality uses the fact that $d \leq 2^{\log N^{(1-\epsilon)}} \leq N$. We thus have

$$\log(B) \leq \log(\alpha+2) + L + \log\log(2N). \tag{12}$$

Substituting equation 12 to equation 11, we get

$$L \geq \log\left(\frac{\log N - \log(\alpha+2) - L - \log\log(2N)}{(\log N)^{1-\epsilon}}\right) \tag{13}$$

Now, if $L > \log\log 2N$, we are done. Otherwise, if $L \leq \log\log(2N)$, then we can substitute this to equation 13 to get

$$\begin{aligned} L &\geq \log\left(\frac{\log N - \log(\alpha+2) - 2\log\log(2N)}{(\log N)^{1-\epsilon}}\right) \\ &= \log\left(\log N - \log(\alpha+2) - 2\log\log 2N\right) - (1-\epsilon)\log\log N \end{aligned} \tag{14}$$

We now claim that first term in equation 14 satisfies the following:

$$\log\left(\log N - \log(\alpha+2) - 2\log\log(2N)\right) \geq (1 - \frac{\epsilon}{2})\log\log N. \tag{15}$$

To see this, note that, for sufficiently large enough N, the following holds:

$$\frac{\log N}{2} \geq \log(\alpha+2) + 2\log\log(2N),$$

hence, we get

$$\log\left(\log N - \log(\alpha+2) - 2\log\log(2N)\right) \geq \log\left(\frac{\log N}{2}\right) \geq \log\log N - 1 \geq (1 - \frac{\epsilon}{2})\log\log N.$$

This proves the claim in equation 15. Finally, using equation 15, equation 14 leads to the following:

$$L \geq (1 - \frac{\epsilon}{2})\log\log N - (1-\epsilon)\log\log N = \frac{\epsilon}{2}\log\log N,$$

which still provides the lower bound $L = \Omega(\epsilon \log\log N)$, as desired. □

G.1.2. Lower Bounds for MQAR with $d = \log_2 c$

Next, we present lower bounds for the *mulitple-query associative recall* (MQAR) problem which generalizes the AR problem (Arora et al., 2023a). To this end, we recall the definition of MQAR below.

> Suppose we are given an input sequence $\boldsymbol{u}[0 \cdots 3N-1] \triangleq \{(\boldsymbol{k}_0, \boldsymbol{v}_0, \boldsymbol{q}_0), \ldots, (\boldsymbol{k}_{N-1}, \boldsymbol{v}_{N-1}, \boldsymbol{q}_{N-1})\}$ with each $\boldsymbol{k}_i, \boldsymbol{v}_i, \boldsymbol{q}_i \in C$ is a token drawn from a vocabulary of size $c = |C|$. Our goal is then to check, for each $1 \leq i \leq N-1$, whether there exists $0 \leq j < i$ such that $\boldsymbol{q}_i \equiv \boldsymbol{k}_j$, and if so, output $\boldsymbol{v}_j$.

We now present the following lower bound from (Arora et al., 2024) for the MQAR problem $d = \log_2 c$ to encode all c possible tokens from C using the natural binary encoding, which also holds for JRT input. This is because the result (Theorem F.5) in (Arora et al., 2024) is derived using Lemma 5.1 in (Arora et al., 2024) (degree of multilinear polynomial computed by `BaseConv` in terms of its number of layers) and Lemma 5.2 in (Arora et al., 2024) (degree of multilinear polynomial for the MQAR problem), both of which are independent of the input length N.

Theorem G.7. *A data-independent* `BaseConv` *model needs* $\log(2d)$*-layers to solve* MQAR *with a JRT prompt* $\boldsymbol{u} \in \{0,1\}^{2 \cdot 3N \times d}$ *for the original input* $\boldsymbol{u} \in \{0,1\}^{3N \times d}$ *with* $d = \log_2(c)$.

G.1.3. Lower Bounds for MQAR via the Equality (EQ) Problem

(Arora et al., 2024) also contains lower bounds on the number of layers solving MQAR due to the lower bounds on the *equality problem* (EQ), where we define the equality problem (EQ) as checking whether the two encodings are equal: $\boldsymbol{u}_1 \equiv \boldsymbol{u}_2$ for an input pair $\boldsymbol{u}_1, \boldsymbol{u}_2$ where each $\boldsymbol{u}_i$ is a token drawn from a vocabulary of size $c = |C|$ and embedded in $\{0,1\}^d$.

We next show that any model with JRT prompts solving MQAR also solves EQ.

Proposition G.8. *Any model M_{MQAR} that solves MQAR with JRT prompt also solves EQ using the same number of layers.*

Proof. If there exists a model $\mathrm{M}_{\mathrm{MQAR}}$ that solves MQAR using L layers with JRT prompt, then for an arbitrary input instance for EQ given by $\boldsymbol{u}_1, \boldsymbol{u}_2 \in \mathbb{R}^{2\times d}$, we can produce the following input instance for MQAR: $\boldsymbol{u} := \{(\boldsymbol{u}_1, \mathbb{1}, \boldsymbol{u}_1), (\boldsymbol{u}_2, \mathbb{1}, \boldsymbol{u}_2), (\boldsymbol{u}_1, \mathbb{1}, \boldsymbol{u}_1), (\boldsymbol{u}_2, \mathbb{1}, \boldsymbol{u}_2)\}$ and solve EQ using L layers with $\mathrm{M}_{\mathrm{MQAR}}$ returning $\mathbb{1}$ iff there is a match. □

Due to proposition G.8, we obtain the following corollary.

Corollary G.9. *Any lower bound $\overline{L}$ on the number of layers L of* `BaseConv` *to solving EQ is also a lower bound on the number of layers required for solving* MQAR *with JRT prompts.*

The lower bounds for the EQ problem in (Arora et al., 2024) depends on showing that the polynomial P representing EQ in p-hot encoding has $\deg(P) \geq 2p$, which does not depend on the sequence length (Proposition F.5). Since corollary G.9 also holds in the JRT setting, we inherit the lower following lower bound for `BaseConv` solving MQAR in the p-hot encoding setting, which we recall here for the reader's convenience.

Definition G.10 (p-Hot Encoding)**.** *We define the* p-hot encoding *to be the collection of embeddings for a token $\boldsymbol{x}_t$ with $0 \leq t < c$ such that we express t in base $\sqrt[p]{c}$: $(t_0, .., t_{p-1}) \in [0, \sqrt[p]{c})^p$ and represent each t_i as one hot encoding in $\{0,1\}^{\sqrt[p]{c}}$. That is, we take $d = p \cdot \sqrt[p]{c}$.*

Theorem G.11. *A data-independent* `BaseConv` *model needs at least $\lfloor \log(2p) \rfloor$-layers to solve* MQAR *for a JRT prompt $\boldsymbol{u}^{\mathrm{JRT}} \in \{0,1\}^{2\cdot 3N\times d}$ for the original input $\boldsymbol{u} \in \{0,1\}^{3N\times d}$ in the p-hot encoding setting, where $d = p \cdot \sqrt[p]{c}$.*

G.2. Recurrent Models and Set Disjointness

In this section, we will provide upper bounds on the class of recurrent models defined in (Arora et al., 2024) solving the set disjointness (SD) problem. First, we recall the definition of recurrent models below.

Definition G.12 (Recurrent Models)**.** *A model $\mathcal{M}$ taking an input $\boldsymbol{u} \in \mathbb{R}^{N\times d}$, where N is the input length and d is the model dimension, is termed a* recurrent model *if its i-th state, representing the output at location i, $\boldsymbol{Z}^i_{\mathcal{M}} \in \mathbb{R}^{\tilde{d}}$, with $\tilde{d}$ denoting the state size, is determined exclusively by the preceding elements of the input $\boldsymbol{u}[0 \ldots i-1]$. The state $\boldsymbol{Z}^i_{\mathcal{M}}$ represents the accumulated information of the model depending on the inputs up to the i-th element, and is distinct from learned parameters that are static with respect to the input sequence.*

Specifically, $\boldsymbol{Z}^i_{\mathcal{M}}(\boldsymbol{u}) = \phi(\boldsymbol{u}[0 \ldots i-1])$, indicating that the state is a function of the input history but not of the entire input sequence simultaneously. Moreover, we can express this as:

$$\boldsymbol{Z}^i_{\mathcal{M}}(\boldsymbol{u}) = f^i_{\mathcal{M}}(\boldsymbol{Z}^{i-1}_{\mathcal{M}}, \boldsymbol{u}[i]), \tag{16}$$

for a sequence of functions $\{f^i_{\mathcal{M}}\}_{i\in[N]}$, where each function is tailored to evolve the state based on the immediate past state and the current input.

Remark G.13. *Note that definition G.12 excludes models that inherently require the entire input sequence for computation at any state, such as those based on non-causal convolutional operations over the full input.*

Remark G.14. *Given sets $A, B \subseteq \{0,1\}^n$, the* set disjointness *(SD) problem seeks to check whether A and B are disjoint, that is, $A \cap B = \emptyset$. First, we clarify the format of the input $\boldsymbol{u} \in \{0,1\}^{N\times(n+1)}$ for the set-disjointness problem with $N = |A| + |B| + 1$. The rows of the input $\boldsymbol{u} \in \{0,1\}^{N\times(n+1)}$ correspond to elements in A and B. That is, $\boldsymbol{u}[i, 0:n-1] \in A \cup B \cup \{\mathbf{0}^n\}$, where $\{[\mathbf{0}^n :: 1]\}$ is a separator element which separates the contiguously placed (in any arbitrary order) elements of each set with the last entry of non-separator rows equal to 0.*

Theorem G.15. *For any recurrent model $\mathcal{M}$, there exists a function of the input history $\boldsymbol{Z}^i_{\mathcal{M}}(\boldsymbol{u}^{\text{JRT}}) = \phi(\boldsymbol{u}^{\text{JRT}}[0 \dots i-1])$ that solves the set disjointness problem with $\boldsymbol{Z}^{2N}_{\mathcal{M}}$ of size $\mathcal{O}(n \cdot \min\{|A|, |B|\})$ for the* JRT *prompt $\boldsymbol{u}^{\text{JRT}} \in \{0,1\}^{2N \times (n+1)}$ of the input $\boldsymbol{u} \in \{0,1\}^{N \times (n+1)}$ for the set-disjointness problem.*

Proof. Given a JRT prompt $\boldsymbol{u}^{\text{JRT}} \in \{0,1\}^{2N \times (n+1)}$ corresponding to the input for the set-disjointness problem, for a recurrent model $\mathcal{M}$, we define the state $\boldsymbol{Z}^i_{\mathcal{M}}$ in Algorithm 3.

Algorithm 3 Recurrent Model for Set Disjointness

Require: an input $\boldsymbol{u}^{\text{JRT}} \in \{0,1\}^{2N \times (n+1)}$ for the set-disjointness problem
Ensure: state size $\boldsymbol{Z}^{2N-1}_{\mathcal{M}}$.

```
1:  firstSeparator ← False
2:  secondSeparator ← False
3:  smallFirst ← False
4:  for i ← 0 to 2N − 1 do
5:      if u^JRT[i, n] = 1 then
6:          if firstSeparator = False then
7:              firstSeparator ← True
8:              if i ≤ ⌊N/2⌋ then
9:                  smallFirst ← True
10:             end if
11:         else
12:             secondSeparator ← True
13:         end if
14:     else
15:         if firstSeparator = True then
16:             if smallFirst = True then
17:                 if secondSeparator = False then
18:                     if i ≥ N then
19:                         Add u^JRT[i, :] to Z^i_M
20:                     end if
21:                 else
22:                     if there exists j s.t. u^JRT[i, :] = Z^{i−1}_M[j, :] then
23:                         Z^{i−1}_M[j, n] = 1
24:                     end if
25:                 end if
26:             else
27:                 if secondSeparator = False then
28:                     if i ≤ N then
29:                         Add u^JRT[i, :] to Z^i_M
30:                     else
31:                         if there exists j s.t. u^JRT[i, :] = Z^{i−1}_M[j, :] then
32:                             Z^{i−1}_M[j, n] = 1
33:                         end if
34:                     end if
35:                 end if
36:             end if
37:         end if
38:     end if
39: end for
40: for all j s.t. Z^{i−1}_M[j, n] = 1 do
41:     return Z^{i−1}_M[j, 0 : n − 1].
42: end for
```

Semantically, we take a JRT input $\boldsymbol{u}^{\text{JRT}} \in \{0,1\}^{2N \times (n+1)}$ for the set-disjointness problem, and find the first separator (lines 5 to 9). If the index i of the first separator is less than or equal to $\lfloor \frac{N}{2} \rfloor$ (line 8), then we know that the smaller set is placed before the larger set. Otherwise, the smaller set is placed later (see Figure 4).

Either way, we want to store the smaller set and compare it against the larger set for intersections. To this end, if the smaller set comes first (line 16), then we continue until the beginning of the repeated input (line 18) and collect the smaller set (line 19), which we then use after we encounter the second separator (lines 22 to 23) to compare against the larger set. If

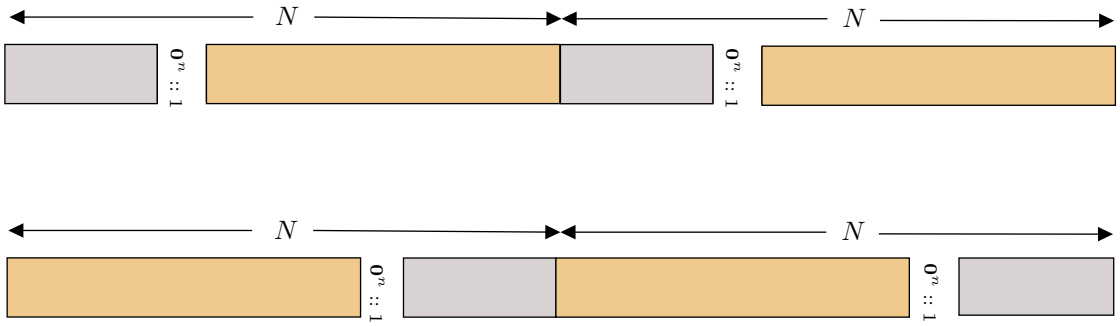

Figure 4: Placement of the smaller set is determined by when we first encounter the separator.

the smaller set comes second (lines 26 to 32), then after the first separator, we collect the smaller set (lines 28 to 29) and compare it against the larger set that comes right after (lines 30 to 32).

For comparison (lines 40 to 41), we use the separator flag at the end. Recall that non-separator elements of the input have 0 in the separator flag index, and thus, so do the elements from the smaller set collected in the state $\boldsymbol{Z}_{\mathcal{M}}$. When comparing against the elements from the larger set, we simply set the flag to 1 for an element that is in the intersection of two sets.

Now, we examine the space requirement for the state $\boldsymbol{Z}_{\mathcal{M}}$ of the model $\mathcal{M}$. Note that we only add an element to $\boldsymbol{Z}_{\mathcal{M}}$ in lines 19 and 29. In both cases, the elements are from the smaller set, and thus, $|\boldsymbol{Z}_{\mathcal{M}}| = \min\{|A|, |B|\}$. Moreover, each element in A and B is of size n, and thus, we can conclude that the model $\mathcal{M}$ with state $\boldsymbol{Z}_{\mathcal{M}}$ can solve the set-disjointness problem with JRT input in $\mathcal{O}(n \cdot \min\{|A|, |B|\})$. □

G.3. Based Solving SD

In this section, we will show that `Based` can solve the set disjointness problem with JRT inputs. Specifically, this section implements Algorithm 3 in the `Based` architecture. Recall here that the `Based` model combines two layer types: `BaseConv` (see definition G.1) and `LinearAttention` defined below.

Definition G.16 (Linear Attention with Kernels)**.** *Given an input sequence $\boldsymbol{u} \in \mathbb{R}^{N \times d}$, where N is the sequence length and d is the model dimension, kernel projections*[10] $\texttt{Projection}_q, \texttt{Projection}_k \in \mathbb{R}^{d \times f}$, $\texttt{Projection}_v \in \mathbb{R}^{d \times d}$, *where f is the feature dimension, the* `LinearAttention` *layer computes the following:*

$$\boldsymbol{z}^{\texttt{LinearAttention}} := \left(\boldsymbol{Q}\,\boldsymbol{K}^\top\right)\boldsymbol{V} \in \mathbb{R}^{N \times d}, \tag{17}$$

where $\boldsymbol{Q} := \texttt{Projection}_q(\boldsymbol{u}), \boldsymbol{K} := \texttt{Projection}_k(\boldsymbol{u}), \boldsymbol{V} := \texttt{Projection}_v(\boldsymbol{u})$.

G.3.1. SD WITH LINEARATTENTION

We first show that with appropriate placement of the two sets, we can solve the set disjointness problem using a class of kernel maps defined below.

Definition G.17 (IP-Kernel)**.** *We define the* IP-Kernel *to be the kernel map $\phi_{\epsilon,f} : \mathbb{R}^d \to \mathbb{R}^f$ that takes elements from $[c]$ to $\mathbb{R}^f$ so that, for any $x, y \in [c]$, we have*

$$\langle \phi_{\epsilon,f}(x), \phi_{\epsilon,f}(y) \rangle = 1 \text{ if } x = y \text{ and } |\langle \phi_{\epsilon,f}(x), \phi_{\epsilon,f}(y) \rangle| \leq \epsilon \text{ otherwise.}$$

That is, an IP-kernel projects elements from the universal set $[c]$ so that the inner products are approximately orthogonal. Note that the feature dimension f is dependent on the tolerance ϵ.

We now show that if there exists an IP kernel with small enough ϵ, then it can be used to solve the set-disjointness problem with a `Linear Attention` layer followed by an MLP layer.

Proposition G.18. *Given an input $\boldsymbol{u} \in \mathbb{R}^{N \times d}$ encoding the input (A, B) to the set-disjointness problem (SD) on sets $A, B \subseteq [c]$, there exists a* `Linear Attention` *(+ MLP) layer with state space $O(df)$ that solves the set disjointness*

[10] By kernel projections of a matrix $\boldsymbol{u} \in \mathbb{R}^{m \times n}$, we mean applying some kernel map $\phi : \mathbb{R}^{N \times d} \to \mathbb{R}^{N \times f}$ to each row of $\boldsymbol{u}$.

problem for $\boldsymbol{u} \in \mathbb{R}^{N\times d}$ with the IP kernel $\phi_{\epsilon,f}$ applied on $\boldsymbol{Q}, \boldsymbol{K}$ for $\epsilon = \frac{1}{3|A|}$.[11]

Proof. We first define the keys and queries along with the values for the `Linear Attention` layer as follows:

$$\boldsymbol{Q}[i,:] = \boldsymbol{K}[i,:] = \phi_{\epsilon,f}(\boldsymbol{u}[i,:]) \text{ and } \boldsymbol{V}[i,j] := \begin{cases} 1 & \text{if } i < |A| \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N\times f}$ and $\boldsymbol{V} \in \mathbb{R}^{N\times d}$.

$$\begin{aligned}\left(\boldsymbol{Q}\,\boldsymbol{K}^\top\right)[i,j] &:= \boldsymbol{Q}[i,:]\boldsymbol{K}^\top[:,j] \\ &= \langle \boldsymbol{Q}[i,:], \boldsymbol{K}[j,:]\rangle \\ &= \langle \phi_{\epsilon,f}(\boldsymbol{u}[i,:]), \phi_{\epsilon,f}(\boldsymbol{u}[j,:])\rangle\end{aligned}$$

Next, the key-query product yields the following

$$\begin{aligned}\boldsymbol{z}^{\texttt{LinearAttention}}[i,j] &:= \left(\boldsymbol{Q}\,\boldsymbol{K}^\top\right)[i,:]\boldsymbol{V}[:,j] \\ &= \sum_{k=0}^{N-1} \left(\boldsymbol{Q}\,\boldsymbol{K}^\top\right)[i,k] \cdot \boldsymbol{V}[k,j] \\ &= \sum_{k=0}^{N-1} \langle \phi(\boldsymbol{u}[i,:]), \phi(\boldsymbol{u}[k,:])\rangle \cdot \boldsymbol{V}[k,j] \\ &= \sum_{k<|A|} \langle \phi_{\epsilon,f}(\boldsymbol{u}[i,:]), \phi_{\epsilon,f}(\boldsymbol{u}[k,:])\rangle \\ &=: \rho_i,\end{aligned}$$

where the second-last equality follows from the definition of $\boldsymbol{V}$ and we can specify ρ_i as follows:

$$\rho_i = 1 \pm \epsilon \cdot |A| \text{ if there exists } k \in [0 \cdots |A|-1] \text{ s.t. } \boldsymbol{u}[k,:] \equiv \boldsymbol{u}[i,:], \text{ and otherwise, } \rho_i \leq \epsilon|A|. \tag{18}$$

For the MLP layer, we define the following parameters (see Definition G.3 for notation):

$$\boldsymbol{W}^1 = \boldsymbol{I}_{d\times d}, \quad \boldsymbol{B}^1_{\text{MLP}} := -\frac{1}{3}\mathbf{1}_{N\times d}, \quad \boldsymbol{W}^2_{\text{MLP}} = \boldsymbol{I}_{d\times d}, \quad \boldsymbol{B}^2_{\text{MLP}} = \mathbf{0}_{N\times d}$$

Next, we note that for $0 \leq \ell < N$ and $0 \leq j < d$:

$$\begin{aligned}\boldsymbol{y}[\ell,j] &:= \left(\boldsymbol{z}^{\texttt{LinearAttention}}\boldsymbol{W}^1_{\text{MLP}} + \boldsymbol{B}^1_{\text{MLP}}\right)[\ell,j] \\ &= \left(\boldsymbol{z}^{\texttt{LinearAttention}} - \frac{1}{3}\mathbf{1}_{|B|\times d}\right)[\ell,j] \\ &= \left(\rho_\ell - \frac{1}{3}\right).\end{aligned}$$

We now use the fact that $\epsilon \leq \frac{1}{3|A|}$ to get bounds on the above. To this end, for $0 \leq \ell < N$, due to equation 18, if there exists $k \in [0 \cdots |A|-1]$ such that $\boldsymbol{u}[k,:] \equiv \boldsymbol{u}[\ell,:]$, we have

$$\boldsymbol{y}[\ell,j] = \left(\rho_\ell - \frac{1}{3}\right) := \left((1 \pm \epsilon \cdot |A|) - \frac{1}{3}\right) \in \left[\frac{2}{3}, \frac{4}{3}\right] - \frac{1}{3} = \left[\frac{1}{3}, 1\right]$$

Otherwise, if there is no match, then we have

$$\boldsymbol{y}[\ell,j] = \left(\rho_\ell - \frac{1}{3}\right) \leq \epsilon \cdot |A| - \frac{1}{3} \leq \frac{1}{3} - \frac{1}{3} \leq 0.$$

[11] Our notion of 'solves' is a bit non-standard so we clarify it here. If $\boldsymbol{z} \in \mathbb{R}^{N\times d}$ is the output then it encodes the result as follows. If the ith element in B appears in A then $\boldsymbol{z}[|A|+i,:]$ has all entries in $\left[\frac{1}{3}, 1\right]$, otherwise it is $\mathbf{0}^d$. If we want a single value as an answer (since SD has a Boolean output) we can apply $O(1)$ `BaseConv` layers on $\boldsymbol{z}$ to sum up all the values in the last $|B|$ rows of $\boldsymbol{z}$. Then if $A \cap B \neq \emptyset$ then this value is at least $\frac{d}{3}$, otherwise it is 0.

We then get the final output as

$$\boldsymbol{z} := \mathrm{ReLU}(\boldsymbol{y})\boldsymbol{W}^2_{\mathrm{MLP}} + \boldsymbol{B}^2_{\mathrm{MLP}} = \mathrm{ReLU}(\boldsymbol{y}),$$

which reduces to

$$\boldsymbol{z}[\ell, j] \in \left[\frac{1}{3}, 1\right] \text{ if there exists } k \in [0 \cdots |A| - 1] \text{ such that } \boldsymbol{u}[k, :] \equiv \boldsymbol{u}[i, :], \text{ and } 0 \text{ otherwise.}$$

Therefore, the last $|B|$ rows of the output $\boldsymbol{z}$ will have non-zero values if and only if $A \cap B \neq \phi$. Finally, the claim on $O(df)$ space follows from the well-known recurrent view of `LinearAttention` (see equation 2).[12] □

G.3.2. Realization of IP Kernels

In this section, we will provide some instances of realizing the IP kernels from Definition G.17.

Exponential Kernels. The first IP-kernel that we define is the exponential kernel $\phi^{\exp} : \mathbb{R}^d \to \mathbb{R}^f$ such that for any $x, y \in [c]$, we have

$$\langle \phi(x), \phi(y) \rangle = \exp\left(\langle x, y \rangle\right),$$

where x and y are encoding of the corresponding elements of $[c]$ in $\{-1, 1\}^d, d = O(\log(c))$ with large enough distance[13]. If $x = y$, we have

$$\langle \phi(x), \phi(y) \rangle = \langle \phi(x), \phi(x) \rangle$$

$$= \exp\left(\langle x, x \rangle\right) = \exp\left(\sum_{i \in [d]} x_i^2\right) = \exp\left(\sum_{i \in [d]} 1\right) = \exp(d).$$

Next, if $x \neq y$, we instead have

$$0 < \langle \phi(x), \phi(y) \rangle = \exp\left(\langle x, y \rangle\right) \leq \exp\left(\gamma \cdot d\right)$$

for some $\gamma < 1$ as the code has constant relative distance. Here, we want the match $\exp(d)$ to be large enough. That is, we want

$$\frac{\exp(d)}{\exp(\gamma \cdot d)} \gg c$$

So, we want to pick d large enough so that

$$(1 - \gamma) \cdot d \gg \ln c.$$

Data-Dependent Kernels. Here, we define the kernel ϕ based on the smaller set A. We start by letting $d := |A| + \log c$ so that we define the embeddings as

$$\begin{aligned} \phi &: [c] \to \mathbb{R}^{|A| + \log c} \\ A \ni a &\mapsto \begin{bmatrix} \boldsymbol{e}_a & \boldsymbol{0}^{\log c} \end{bmatrix} \\ A \not\ni b &\mapsto \begin{bmatrix} \boldsymbol{0}^{|A|} & \boldsymbol{B}_b \end{bmatrix} \end{aligned} \tag{19}$$

where $\boldsymbol{e}_a \in \{0, 1\}^{|A|}$ is the 1-hot encoding of the element a in A and $\boldsymbol{B}_b$ is the natural binary encoding in $[c]$ on the element b. Using this kernel ϕ, we achieve orthogonality:

$$\langle \phi(x), \phi(y) \rangle = \delta_{xy}.$$

That is, we have the tolerance $\epsilon = 0$ with feature dimension $f = |A| + \log_2 c$.

[12] To incorporate the MLP part, note that as soon as each row of $\boldsymbol{z}^{\texttt{LinearAttention}}$ is generated, we can generate the output of the corresponding row in $\texttt{MLP}(\boldsymbol{z}^{\texttt{LinearAttention}})$ with $O(d)$ space by noting that MLP operates independently on each row of its input.

[13] Specifically, we will need to use well-known construction of Binary codes with constant rate and constant relative distance (Guruswami et al., 2019).

Randomized Kernels. We can also define a random kernel map

$$\phi : [c] \to \frac{1}{\sqrt{f}}[-1, 1]^f. \tag{20}$$

That is, for each $x \in [c]$, we pick a random vector in $\{-1, 1\}^f$ and normalize it by dividing by $\sqrt{f}$. Here, it is easy to see that for every $x \in [c]$, we have

$$\langle \phi(x), \phi(x) \rangle = \frac{1}{f} \sum_{i \in [f]} 1 = 1.$$

Now, for every $x \neq y$, we can apply known concentration inequalities on Rademacher random variables to get

$$\Pr\left[\langle \phi(x), \phi(y) \rangle > \frac{t}{\sqrt{f}} \right] \leq e^{\frac{-t^2}{2}}.$$

We then pick $t = O(\sqrt{\log c})$ so that over all c^2 pairs, we have

$$\Pr\left[\langle \phi(x), \phi(y) \rangle > \frac{O(\sqrt{\log c})}{\sqrt{f}} \right] < \frac{1}{100c^2}.$$

Then with a union bound on all c^2 pairs, with high probability, we get that ϕ has $\epsilon = \frac{t}{\sqrt{f}}$. We then want the threshold to satisfy the following:

$$t/\sqrt{f} < \frac{1}{3|A|} \implies f = \Omega(|A|^2 \log c).$$

That is, for $\epsilon = \frac{1}{3|A|}$, $f = \Theta(\min\{|A|, |B|\}^2 \log c)$ suffices.

Remark G.19 (Practical Justification)**.** *Empirically, prior works shows a variety of kernels that are competitive with softmax attention quality while using a small amount of space. For instance, Zhang et al. (Zhang et al., 2024) show that either training MLP projections to mimic softmax attention weights or using a 2nd-order Taylor approximation to the softmax-exponential function are two effective kernel function choices. The 2nd-order polynomial is only a high fidelity approximation within a small band of real values, however empirically results in Arora et al. (Arora et al., 2024) suggest that the normalized query-key dot products often fall within this range, resulting in competitive quality with softmax attention. Arora et al. (Arora et al., 2024), Chen et al. (Chen et al., 2021), and others further suggest that combining efficient sparse plus low-rank attentions (e.g., linear attention plus dense, local sliding window attention) further diminishes quality gaps versus full attention.*

G.3.3. Shifts with BaseConv

Next, we will show that we can use `BaseConv` layers to move the smaller set to the start of the sequence. First, based on whether the smaller set is at the start or not, we need to define separate convolution kernels based on the input. To this end, we use the following `BaseConv` model to derive these kernels.

Lemma G.20. *There exists $\left(2N, O(1), (n+1), \left(2N + \frac{N}{2}\right), (n+1)\right)$ − `BaseConv` model that takes in a* JRT *prompt $\boldsymbol{u}^{\text{JRT}} \in \mathbb{R}^{2N \times (n+1)}$ of the input $\boldsymbol{u} \in \mathbb{R}^{N \times (n+1)}$ for the set-disjointness (SD) problem $(A, B) \subseteq \{0, 1\}^n$ and outputs the kernel $\boldsymbol{h}_{\text{shift}}$ that shifts the input $\boldsymbol{u}^{\text{JRT}}$ to get the smaller set at the start of the sequence, where*

$$\boldsymbol{h}_{\text{shift}}(X) := \begin{cases} X^{|A|+1} & \text{if } |A| \geq |B| \\ 1 & \text{otherwise.} \end{cases} \tag{21}$$

Proof. Following the proof of Proposition G.18, we know that it suffices to find the location of the separator to determine the location of the smaller set. More specifically, if the separator is within $\left[0, \frac{N}{2} - 1\right]$ row index range, then we know that the smaller set is at the start, and the kernel being generated is the identity. Otherwise, we generate the kernel $X^{|A|+1}$ which will be used in the proof of Proposition G.21.

We first increase the inner dimension of the JRT input $\boldsymbol{u}^{\text{JRT}} \in \mathbb{R}^{2N \times (n+1)}$ to $\boldsymbol{u}^{JRT}_{\text{inner}} \in \mathbb{R}^{\left(2N + \frac{N}{2}\right) \times (n+1)}$ so that we introduce a zero-block between the first seperator and the start of set B. That is, we have

$$\boldsymbol{u}^{\text{JRT}}_{\text{inner}}[i, :] = \begin{cases} \boldsymbol{u}^{\text{JRT}}[i, :] & \text{if } i < \frac{N}{2} \\ \boldsymbol{0}^{n+1} & \text{if } \frac{N}{2} \leq i < N \\ \boldsymbol{u}^{\text{JRT}}[i - \frac{N}{2}, :] & \text{if } i \geq N. \end{cases}$$

We can achieve this by simply using the *remembering primitive* from (Arora et al., 2024, Definition F.15, Proposition F.13) using a $\left(\left(2N+\frac{N}{2}\right), 8, (n+1), \left(2N+\frac{N}{2}\right), (n+1)\right) - \texttt{BaseConv}$ to remember $\boldsymbol{u}^{\mathrm{JRT}}[\frac{N}{2} : 2N-1, :]$ while applying the identity kernel to preserve $\boldsymbol{u}^{\mathrm{JRT}}[0 : \frac{N}{2}-1, :]$.

We again apply the *remembering primitive* from (Arora et al., 2024, Definition F.15, Proposition F.13) to get

$$\boldsymbol{Y} \leftarrow \texttt{remember}(\boldsymbol{u}^{\mathrm{JRT}}_{\mathrm{inner}}, 0, N, f),$$

using $\left(\left(2N+\frac{N}{2}\right), 8, (n+1), \left(2N+\frac{N}{2}\right), (n+1)\right) - \texttt{BaseConv}$, where f is applied over $\boldsymbol{x} := \boldsymbol{u}^{\mathrm{JRT}}_{\mathrm{inner}}[0 : N-1, :]$, the first N rows of $\boldsymbol{u}^{\mathrm{JRT}}_{\mathrm{inner}}$. That is, we want to remember the last $\left(N+\frac{N}{2}\right)$ rows of $\boldsymbol{u}^{\mathrm{JRT}}_{\mathrm{inner}}$. We define $f := f_2 \circ f_1$, where f_1 is the cumulative sum of the first N rows computed using $(N, O(1), (n+1), N, (n+1)) - \texttt{BaseConv}$ followed by f_2 which is the shifting down by $N-1$ using $(N, 3, (n+1), N, (n+1)) - \texttt{BaseConv}$ (**?**)Propositions F.41 and F.38]arora2024simple. That is, for $i \in [0 : N-1]$, we have

$$f_1(\boldsymbol{x})[i, :] = \sum_{k=0}^{i} \boldsymbol{x}[k, :];$$
$$f_2(f_1(\boldsymbol{x}))[i, :] = f_1(\boldsymbol{x})[i-(N-1), :].$$

For the nth column, we know that for $0 \leq i < N$:

$$\boldsymbol{x}[i, n] = \boldsymbol{u}^{\mathrm{JRT}}_{\mathrm{inner}}[i, n] = \boldsymbol{u}^{\mathrm{JRT}}[i, n] = \begin{cases} 1 & \text{if } |A| \leq |B| \text{ and } i = |A| \\ 0 & \text{otherwise.} \end{cases}$$

This is because if $|A| \leq |B|$, the separator is within $\left[0, \frac{N}{2}-1\right]$ and its nth bit is 1, where $|A| =: i_s \in \left[0, \frac{N}{2}-1\right]$ to be the location of the separator. We then get

$$f_1(\boldsymbol{x})[i, n] = \begin{cases} 1 & \text{if } |A| \leq |B| \text{ and } i \geq i_s \\ 0 & \text{otherwise.} \end{cases}$$
$$f_2(f_1(\boldsymbol{x}))[i, n] = \begin{cases} 1 & \text{if } |A| \leq |B| \text{ and } i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

We can thus characterize the nth column of the output $\boldsymbol{Y} \in \mathbb{R}^{\left(2N+\frac{N}{2}\right) \times (n+1)}$ as follows:

$$\boldsymbol{Y}[i, n] = \begin{cases} 1 & \text{if } |A| \leq |B| \text{ and } i = 0 \\ 0 & \text{if } |A| > |B| \text{ and } i = 0 \text{ or } 1 \leq i < N \\ \boldsymbol{u}^{\mathrm{JRT}}[i+\frac{N}{2}, n] & \text{if } i \geq N. \end{cases}$$

We now remember $\boldsymbol{Y}[0 : \frac{N}{2}-1, :]$ while shifting down $\boldsymbol{Y}[\frac{N}{2} : 2N+\frac{N}{2}-1, :]$ by $\frac{N}{2}-1$ (**?**)Proposition F.13 and F.38]arora2024simple to get $\boldsymbol{Y}'$ such that:

$$\begin{aligned} \boldsymbol{Y}'[i, :] &= \begin{cases} \boldsymbol{Y}[i, :] & \text{if } i < \frac{N}{2} \\ \boldsymbol{Y}[i-\frac{N}{2}, :] & \text{if } i \geq \frac{N}{2} \end{cases} \\ &= \begin{cases} \boldsymbol{Y}[i, :] & \text{if } i < \frac{N}{2} \\ \boldsymbol{u}^{\mathrm{JRT}}[i, :] & \text{if } \frac{N}{2} \leq i < 2N-1 \\ \boldsymbol{0}^n & \text{otherwise.} \end{cases} \end{aligned}$$

Focusing on the nth column, we see that we get for $0 \leq i < N$:

$$\boldsymbol{Y}'[i, n] = \begin{cases} 1 & \text{if } |A| \leq |B| \text{ and } i = 0 \text{ or } |A| > |B| \text{ and } i = |A| \\ 0 & \text{otherwise} \end{cases}.$$

Or equivalently

$$\boldsymbol{Y}'[0:N-1,n] = \begin{cases} \boldsymbol{e}_0 & \text{if } |A| \leq |B| \\ \boldsymbol{e}_{|A|} & \text{if } |A| > |B|. \end{cases},$$

which is exactly what we need as the shift kernel $\boldsymbol{h}_{\text{shift}}$. A schematic representation of this process is provided in Figure 5. The final claim on the overall parameters follows from the fact that we can 'stack' `BaseConv` layers with the same internal dimension (Arora et al., 2024).

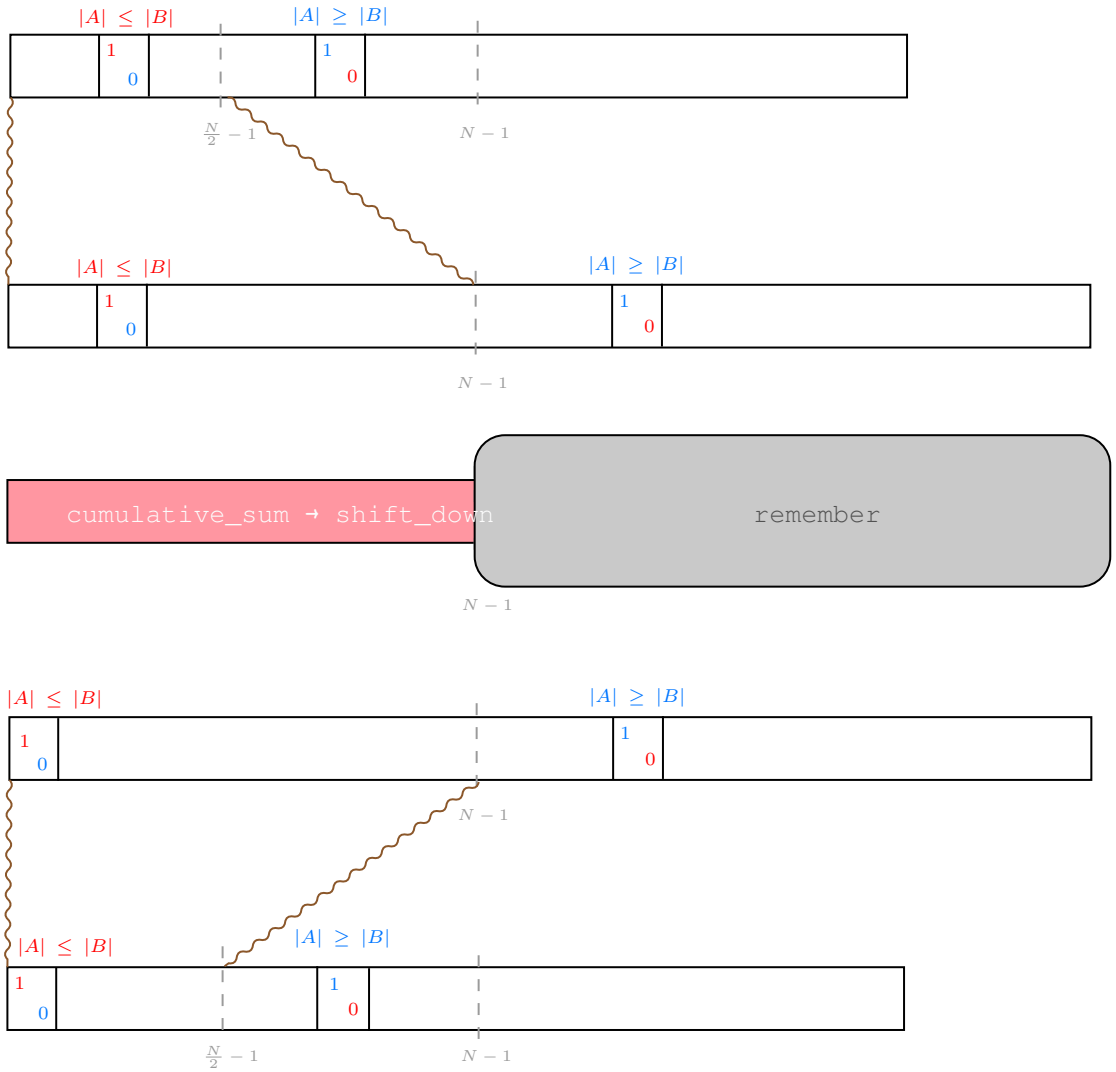

Figure 5: Schema for getting input-dependent shift kernels for the set disjointness (SD) problem.

□

We now use the kernels from Lemma G.20 to do the appropriate shift.

Proposition G.21. *Given a* JRT *prompt* $\boldsymbol{u}^{\text{JRT}} \in \mathbb{R}^{2N\times(n+1)}$ *of the input* $\boldsymbol{u} \in \mathbb{R}^{N\times(n+1)}$ *for the set-disjointness (SD) problem* $(A, B) \subseteq \{0,1\}^n$*, there exist* $O(1)$ input-dependent `BaseConv` *layers that can rearrange the input so that the smaller set out of* A *and* B *is placed at the start of the sequence.*

Proof. The input $\boldsymbol{u} \in \{0,1\}^{N\times(n+1)}$ is formatted as in Remark G.14. In the first case where A is the smaller set, we do not need to change the input. Let $\boldsymbol{s} := [\mathbf{0}^n :: 1]$ be the separator, then we want:

$$\boldsymbol{u}^{\text{JRT}} \equiv [\longleftarrow A \longrightarrow \quad \boldsymbol{s} \quad \longleftarrow B \longrightarrow \quad | \quad \longleftarrow A \longrightarrow \quad \boldsymbol{s} \quad \longleftarrow B \longrightarrow]$$

Otherwise, if $|B| \leq |A|$, we want to shift the input so that $|B|$ comes at the start of the input sequence in the JRT prompt $\boldsymbol{u}^{\text{JRT}}$. To this end, we want to add a separator between after the first copy of the input ends. For this purpose, we can keep the first copy as is, and operate on the duplicate A by shifting it down by 1 and adding a separator at the start of the second input. We thus apply the $\texttt{remember}(\boldsymbol{u}^{\text{JRT}}_{\texttt{shift_up}}, N, N + |A|, f)$ primitive (**?**)Definition F.15]arora2024simple with 8 layers of `BaseConv` where f is any function that maps $(A, \boldsymbol{s}) \mapsto (\boldsymbol{s}, A)$, so that we get

$$\boldsymbol{u}^{\text{JRT}} \equiv \left[\longleftarrow A \longrightarrow \quad \boldsymbol{s} \quad \longleftarrow B \longrightarrow \quad \Big| \quad \boldsymbol{s} \quad \longleftarrow A \longrightarrow \quad \longleftarrow B \longrightarrow\right]$$

Next, we shift up using the $\texttt{shift_up}(\boldsymbol{u}^{\text{JRT}}, |A| + 1)$ primitive (**?**)Proposition C.5]arora2023zoology for `BaseConv` with 3 layers by implementing the kernel $\boldsymbol{h}_{\text{shift}}$ from Lemma G.20. We then get

$$\boldsymbol{u}^{\text{JRT}}_{\texttt{shift_up}} \equiv \left[\longleftarrow B \longrightarrow \quad \boldsymbol{s} \quad \longleftarrow A \longrightarrow \quad \Big| \quad \longleftarrow B \longrightarrow \quad \mathbf{0}^{|A|+1}\right]$$

That is, in both cases, the final output has the smaller set out of A and B at the start of the sequence.

To complete the proof we note that we can do the above in one single model (that uses data dependent convolutions): (1) We add the extra separator after the second set in $\boldsymbol{u}^{\text{JRT}}$ and (2) we do the using the convolution operator in `BaseConv` where we use the convolution kernel computed from Lemma G.20.[14] □

Finally, we can combine Propositions G.18 and G.21 to claim that `Based` can solve SD with JRT-prompting in space $O(\min\{|A|, |B|\})$.

Theorem G.22. *Given a* JRT *prompt* $\boldsymbol{u}^{\text{JRT}} \in \mathbb{R}^{2N \times (n+1)}$ *of the input* $\boldsymbol{u} \in \mathbb{R}^{N \times (n+1)}$ *for the set-disjointness (SD) problem* (A, B)*, there exists a (data dependent)* `Based` *model (*`BaseConv` *+ MLP +* `LinearAttention` *+ MLP)*[15] *that solves the SD problem with space* $O(\min\{|A|, |B|\} \cdot n)$.

Proof. First, we use the `BaseConv` layers from Proposition G.21 to get the smaller set of A and B in $\boldsymbol{u}^{\text{JRT}}$ to the start of the sequence in $\boldsymbol{z}^{\texttt{BaseConv}}$. Next, we reduce $\boldsymbol{z}^{\texttt{BaseConv}}$ using an MLP layer to get $\boldsymbol{z}^{\texttt{BaseConv}}[0 : N-1, :]$ as the input to the `LinearAttention` (+MLP) layer in Proposition G.18 so that we solve the SD problem for the original input $\boldsymbol{u}$. Finally, for the `LinearAttention` layer, we can use the data-dependent IP kernels from equation 19 to get $f = O(\min\{|A|, |B|\}))$, which yields the claimed space usage since we have $d = n$. □

Remark G.23. *We note that we can use the random kernels from equation 20 in Theorem G.22 to get space usage of* $O\left((\min\{|A|, |B|\})^2 \cdot n\right)$ without *using data-dependent IP kernels.*

G.4. GAR and SD

In this section, we introduce the *general associative recall* GAR problem. Recall that the query in the AR problem comes at the end, and thus, the query is compared with all the keys in the input. On the other hand, in MQAR, a query at position i is only compared with keys at positions $j < i$. Moreover, the number of keys and queries in the input are the same for MQAR. Instead, we introduce the following alternate generalization of AR that has all the queries at the end with the number of queries different from the number of keys.

Definition G.24 (GAR)**.** *We are given an input sequence*

$$\boldsymbol{u}[0 \cdots N-1] \triangleq (\boldsymbol{k}_0, \boldsymbol{v}_0), \ldots, (\boldsymbol{k}_{n-1}, \boldsymbol{v}_{n-1}); \boldsymbol{q}_0, \ldots, \boldsymbol{q}_{m-1}, \tag{22}$$

where $K := \{\boldsymbol{k}_i\}_{i=0}^{n-1}, V := \{\boldsymbol{v}_i\}_{i=0}^{n-1}$*, and* $Q := \{\boldsymbol{q}_i\}_{i=0}^{m-1}$*, with each* $\boldsymbol{k}_i, \boldsymbol{v}_i, \boldsymbol{q}_i \in C$ *is a token drawn from a vocabulary of size* $c = |C|$*, and we have* $N = 2n + m$.

Our goal in the general associative recall *(*GAR*) problem is to check, for each* $\boldsymbol{q}_i \in Q$*, whether there exists* $\boldsymbol{k}_j \in K$ *such that* $\boldsymbol{q}_i \equiv \boldsymbol{k}_j$*; if so, output the corresponding value* $\boldsymbol{v}_j$*, and otherwise, output* `Null`.

We will first show that SD reduces to GAR.

Proposition G.25. *Any algorithm* $\mathcal{A}$ *solving* GAR *can also solve* SD.

[14] We also need to only keep the first N rows of the matrix, which we can obtain by zeroing out all the remaining rows using another `BaseConv` layer.

[15] This matches the architecture in our experiments.

Proof. Given an input to the set-disjointness problem (A, B) with $A := \{A_0, \ldots, A_{|A|-1}\}, B := \{B_0, \ldots, B_{|B|-1}\}$, we can construct the following input to the GAR problem:

$$\boldsymbol{u} := (A_0, A_0), \ldots, (A_{|A|-1}, A_{|A|-1}); B_0, \ldots, B_{|B|-1}.$$

Now, we run algorithm $\mathcal{A}$ on $\boldsymbol{u}$, and if for all $q \in Q$, we get `Null`, then we know $A \cap B = \emptyset$, and otherwise, $A \cap B \neq \emptyset$. This solves the set disjointness (SD) problem. □

What we have shown is that GAR is much more general compared to SD. However, we can also show that we can solve GAR under certain conditions if we had access to an algorithm solving SD.

Proposition G.26. *Let $\mathcal{A}_{\mathrm{SD}}$ be an algorithm solving the set disjointness (*SD*) problem. Then, for a vocabulary $\mathcal{C}$ with $|\mathcal{C}| = c$ with values from $[c]$ and represented as $v_j \in \{0,1\}^d$ where $d = \lceil \log_2(c+1) \rceil$ with at most one match for each query, we can solve the* GAR *problem (definition G.24) with d calls to $\mathcal{A}_{\mathrm{SD}}$.*

Proof. Given an input $(\boldsymbol{k}_0, \boldsymbol{v}_0), \ldots, (\boldsymbol{k}_{n-1}, \boldsymbol{v}_{n-1}); \boldsymbol{q}_0, \ldots, \boldsymbol{q}_{m-1}$ to GAR, for each call $\ell \in [d]$ to algorithm $\mathcal{A}_{\mathrm{SD}}$, we construct the inputs to algorithm $\mathcal{A}$ by taking $A := Q, B := K_\ell$ with K_ℓ defined as follows:

$$k_j \in K_\ell \iff v_j[\ell] = 1. \tag{23}$$

That is, we include $k_j \in K_\ell$ iff the ℓ'th bit of v_j is 1.

We now claim that we can solve the MQAR problem given $Q \cap K_\ell$ for all $\ell \in [d]$. To see this, note that if a query $q \in Q$ is not in K, then $q \notin Q \cap K_\ell$ for every $\ell \in [d]$. We thus output `Null` for these queries.

Otherwise, if $q \in Q \cap K$, then there exists a non-empty set of calls $L \subseteq [d]$ such that $q \in Q \cap K_\ell$ for all $\ell \in L$. We can then extract the ℓ'th bit of v_j, where $q = k_j$. That is, for $q = k_j$, we use equation 23 to get

$$v_j[\ell] = \begin{cases} 1 & \text{if } \ell \in L \\ 0 & \text{otherwise.} \end{cases}$$

This is exactly the value corresponding to the unique matching key k_j for the query q. □

G.4.1. Lower Bound for GAR via SD

In this section, we present a lower bound for solving GAR. For this purpose, we require the following two-way randomized communication complexity[16] lower bound for set-disjointness (SD).

Theorem G.27 ((Håstad and Wigderson, 2007)[17])**.** *The two-way randomized communication complexity of the set disjointness problem with sets $A, B \subseteq [n]$ is $\Omega(\min\{|A|, |B|\})$ bits for $n \geq o(\min\{|A|, |B|\})$.*

Definition G.28 (JR$-p$ Prompts)**.** *For any model $\mathcal{M}$ with input $\boldsymbol{u} \in \mathbb{R}^{N \times d}$, a* JR$-p$ prompt *for input $\boldsymbol{u}$ is the p-times repeated input $\boldsymbol{u}^{\mathrm{JR}-p} \in \mathbb{R}^{pN \times d}$ given by*

$$\boldsymbol{u}^{\mathrm{JR}-p}[i, :] := \boldsymbol{u}[i \mod N, :]$$

Proposition G.29. *Given a* JR$-p$ prompt *$\boldsymbol{u}^{\mathrm{JR}-p} \in \{0,1\}^{pN \times d}$ for input $\boldsymbol{u} \in \{0,1\}^{N \times d}$ to the* GAR *problem, any recurrent model $\mathcal{M}_{\mathrm{GAR}}$ (definition G.12) solving* GAR *requires $\max_i \left| \boldsymbol{Z}^i_{\mathcal{M}_{\mathrm{GAR}}} \right|$ to be at least $\Omega\left(\frac{\min\{|A|,|B|\}}{p}\right)$-bits.*

Proof. We first take the input $\boldsymbol{u} \in \{0,1\}^{N \times d}$ to the GAR problem and design a two-way communication protocol for solving GAR given access to the reccurrent model $\mathcal{M}_{\mathrm{GAR}}$. To this end, Alice with their access of key-value part generates her part of the input:

$$\boldsymbol{u}_{\mathrm{Alice}} := (k_0, v_0), \ldots, (k_{n-1}, v_{n-1}) \tag{24}$$

[16] Here, in contrast to one-way randomized communication protocol in appendix G.1.1, both Alice and Bob are allowed to send messages to each other.

[17] (Håstad and Wigderson, 2007) provides a lower bound of n for $|A| = |B|$. However, we can extend it to Theorem G.27 by reducing the $\min\{|A|, |B|\}$ subset to the equal sized set by picking a hard distribution where both sets are of size $\min\{|A|, |B|\}$ and then adding "extra" elements to only one of them to get a larger set (i.e., one can increase the universe size by these extra elements to get the desired lower bound).

of the input for GAR (without the queries), and Bob with their access of the query part generates the following;

$$\boldsymbol{u}_{\text{Bob}} := q_0, \ldots, q_{m-1} \tag{25}$$

of the input for GAR (without the key-value pairs) as in equation 22. That is, the concatenation $\boldsymbol{u}_{\text{Alice}} :: \boldsymbol{u}_{\text{Bob}} \equiv \boldsymbol{u}$ in equation 22. We then have

$$\underbrace{\boldsymbol{u}_{\text{Alice}} :: \boldsymbol{u}_{\text{Bob}} :: \cdots :: \boldsymbol{u}_{\text{Alice}} :: \boldsymbol{u}_{\text{Bob}}}_{p-\text{times}} \equiv \boldsymbol{u}^{\text{JR}-p}, \tag{26}$$

the corresponding JR$-p$ prompt for the input $\boldsymbol{u}$ to the GAR problem. We now claim that the following protocol (algorithm 4) is equivalent to running the recurrent model $\mathcal{M}_{\text{GAR}}$ on the JR$-p$ prompt $\boldsymbol{u}^{\text{JR}-p}$:

Algorithm 4 Communication Protocol for GAR

Require: A recurrent model $\mathcal{M}_{\text{GAR}}$ solving GAR along with the inputs $\boldsymbol{u}_{\text{Alice}}, \boldsymbol{u}_{\text{Bob}}$ from 24 and 25.
Ensure: $\mathcal{M}_{\text{GAR}}(\boldsymbol{u}^{\text{JR}-p})$.

```
for i ← 0 to p − 1 do
    for j ← 0 to 2n − 1 do
        Z^{i·N+j}_{M_GAR} ← f^{i·N+j}_M(Z^{i·N+j−1}_{M_GAR}, u_Alice[j])        ▷ Z^0_{M_GAR} ← u_Alice[0, :]
    end for
    Alice sends Z^{i·N+2n−1}_{M_GAR} to Bob
    for j ← 0 to m − 1 do
        Z^{i·N+2n+j}_{M_GAR} ← f^{i·N+j}_M(Z^{i·N+2n+j−1}_{M_GAR}, u_Bob[j])
    end for
    Bob sends Z^{i·N+m−1}_{M_GAR} to Alice
end for
```

The equivalency of this protocol with running the model $\mathcal{M}_{\text{GAR}}$ follows from equation 26.

Next, consider an instance $\boldsymbol{u}^{\text{SD}} := (A, B)$ of the set-disjointness problem with $A, B \subseteq [n]$ and $|A| + |B| = N$, where $A := \{A_0, \ldots, A_{|A|-1}\}, B := \{B_0, \ldots, B_{|B|-1}\}$. Due to proposition G.25, we know that we can generate an equivalent input $\boldsymbol{u}$ for GAR given an input $\boldsymbol{u}^{\text{SD}}$ to the SD problem, whence we can generate inputs for Alice and Bob as in equation 24 and equation 25. Applying algorithm 4 then solves the GAR problem for $\boldsymbol{u}$, and consequently, the SD problem for $\boldsymbol{u}^{\text{SD}}$. Here, the total number of bits that are communicated in this protocol is

$$T_{\text{bits}} := \sum_{i=0}^{p-1} \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{i\cdot N+2n-1}\right| + \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{i\cdot N+m-1}\right|.$$

Now, if T_{bits} is $o(\min\{|A|, |B|\})$ bits, we have shown that a two-way communication protocol exists for solving the set-disjointness (SD) that uses $o(\min\{|A|, |B|\})$ communication complexity. However, this contradicts theorem G.27. Thus, we have $T_{\text{bits}} \geq \Omega\left(\min\{|A|, |B|\}\right)$.

Finally, note that we have

$$\begin{aligned}
p \cdot 2 \max_k \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{k}\right| &= \sum_{i=0}^{p-1} 2 \max_k \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{k}\right| \\
&\geq \sum_{i=0}^{p-1} \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{i\cdot N+2n-1}\right| + \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{i\cdot N+m-1}\right| \\
&\geq \Omega\left(\min\{|A|, |B|\}\right). \\
\implies \max_k \left|\boldsymbol{Z}_{\mathcal{M}_{\text{GAR}}}^{k}\right| &\geq \Omega\left(\frac{\min\{|A|, |B|\}}{2p}\right).
\end{aligned}$$

This concludes the proof. □

Table 15: JRT-RNN Training Settings. For hybridizing the three layer types – gated convolutions, sliding window, and linear attention – we use linear attention at layers $\{2, 7, 12, 17, 22, 27, 32\}$ and sliding window at layers $\{3, 8, 13, 18, 23, 28, 33\}$, with gated convolution layers elsewhere. We did not tune the layer orderings and proportions.

	356M	1.3B
Optimizer	Adam	
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$	
Optimizer eps	$1e-8$	
Precision	BFloat16	
Encoder region length	1024	
Masked language modeling probability	15%	
MLM loss scale	0.25	
NTP loss scale	1.00	
Warmup	1%	
Learning rate decay	Cosine	
Learning rate (min, base)	8e-5, 8e-4	
Global batch size	256	
Weight decay	0.1	
Num Layers	26	36
Hidden Size	1024	1792
MLP Activation	SwiGLU	
MLP Width	2	
Num. Linear Attn Layers	5	7
Num. Linear Attn Heads	16	
Taylor Feature Dimension	16	
Linear Attn Positional Encodings	None	
Num. Sliding Window Layers	5	7
Sliding Window Size	64	16
Sliding Window Heads	16	
Sliding Window Positional Encodings	Rotary	
Num. `BaseConv` Layers	17	22
`BaseConv` Projection Expansion Factor	4	
`BaseConv` Filter Size	3	
`BaseConv` Activation	SiLU	

Table 16: Based Training Settings. For hybridizing the three layer types – gated convolutions, sliding window, and linear attention – we use linear attention at layers $\{2, 7, 12, 17, 22, 27, 32\}$ and sliding window at layers $\{3, 8, 13, 18, 23, 28, 33\}$, with gated convolution layers elsewhere. We did not tune the layer orderings and proportions.

	363M	1.4B
Optimizer	Adam	
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$	
Optimizer eps	$1e-8$	
Precision	BFloat16	
Warmup	1%	
Learning rate decay	Cosine	
Learning rate (min, base)	8e-5, 8e-4	
Global batch size	256	
Weight decay	0.1	
Num Layers	27	36
Hidden Size	1024	1792
MLP Activation	SwiGLU	
MLP Width	2	
Num. Linear Attn Layers	5	7
Num. Linear Attn Heads	16	
Taylor Feature Dimension	16	
Linear Attn Positional Encodings	None	
Num. Sliding Window Layers	5	7
Sliding Window Size	128	
Sliding Window Heads	16	
Sliding Window Positional Encodings	Rotary	
Num. `BaseConv` Layers	17	22
`BaseConv` Projection Expansion Factor	4	
`BaseConv` Filter Size	3	
`BaseConv` Activation	SiLU	

Table 17: Mamba Training Settings

	358M	1.3B
Optimizer	Adam	
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$	
Optimizer eps	$1e-8$	
Precision	BFloat16	
Warmup	1%	
Learning rate decay	Cosine	
Learning rate (min, base)	8e-5, 8e-4	
Global batch size	256	
Weight decay	0.1	
Num Layers	46	
Hidden Size	1024	2048
RMSNorm	True	
Norm Epsilon	$1e-5$	
Dt State	16	
Dt (Min, Max)	$(0.001, 0.1)$	
Dt Init. Strategy	Random	
Dt Init. Floor	$1e-4$	
Dt Scale	1.0	
Dt Softplus	True	
Projection Expansion Factor	2	
Short Conv Filter Size	4	

Table 18: Attention Training Settings

	360M	1.3B
Optimizer	Adam	
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$	
Optimizer eps	$1e-8$	
Precision	BFloat16	
Warmup	1%	
Learning rate decay	Cosine	
Learning rate (min, base)	8e-5, 8e-4	
Global batch size	256	
Weight decay	0.1	
Num Layers	24	36
Hidden Size	1024	1680
Num Heads	16	24
RMSNorm	True	
MLP Bias	False	
Flash Attn	True	
Rotary Emb. Fraction	0.5	
MLP Activation	SwiGLU	
MLP Width	4	