Repeat After Me: Transformers are Better than State Space Models at Copying

Transformers are Better than State Space Models at Copying

Samy Jelassi [1] **David Brandfonbrener** [2] **Sham M. Kakade** [2 3] **Eran Malach** [2]

Abstract

Transformers are the dominant architecture for sequence modeling, but there is growing interest in models that use a fixed-size latent state that does not depend on the sequence length, which we refer to as "generalized state space models" (GSSMs). In this paper we show that while GSSMs are promising in terms of inference-time efficiency, they are limited compared to transformer models on tasks that require copying from the input context. We start with a theoretical analysis of the simple task of string copying and prove that a two layer transformer can copy strings of exponential length while GSSMs are fundamentally limited by their fixed-size latent state. Empirically, we find that transformers outperform GSSMs in terms of efficiency and generalization on synthetic tasks that require copying the context. Finally, we evaluate pretrained large language models and find that transformer models dramatically outperform state space models at copying and retrieving information from context. Taken together, these results suggest a fundamental gap between transformers and GSSMs on tasks of practical interest.

1. Introduction

Transformers (Vaswani et al., 2017) are the workhorse of modern sequence modeling, achieving remarkable performance on a variety of tasks, but they have unavoidable inefficiencies. Specifically, they require $\Omega(L)$ memory[1] and compute to predict the next token of a sequence of length L.

This has spurred a boom in attempts to create architectures that can achieve similar performance as transformers, but with $O(1)$ memory to predict each token. This class of models includes state space models like S4 (Gu et al., 2021) or Mamba (Gu & Dao, 2023), as well as traditional RNN models (Hochreiter & Schmidhuber, 1997) and models that can be trained in parallel like linear attention (Katharopoulos et al., 2020; Choromanski et al., 2020) and parallel RNNs (Bradbury et al., 2016; Peng et al., 2023; Sun et al., 2023). In this paper, we will refer to this entire class of models that use a fixed-size memory as "generalized state space models" or GSSMs (see a formal definition in Section 2).

Recent work has demonstrated impressive performance of GSSMs, but it is not yet clear what these models sacrifice for their improved efficiency, if anything. In this paper, we find that one particular capability that is sacrificed is the ability to retrieve and repeat parts of the input context. As a result, transformers are better than GSSMs at a variety of tasks that require accessing arbitrary parts of the context.

To understand this gap in capabilities, we begin by presenting a theoretical analysis of the copying task[2]. First, we show via construction that a simple transformer model can copy strings of length that is exponential in the number of heads of the transformer. This construction relies on the ability of the transformer to implement a mechanism of "storage" and retrieval of sequences of n tokens (n-grams), where the n-grams are used to track where to copy from. In contrast, we show that, trivially, GSSMs cannot accurately copy strings with more bits than the size of the latent state.

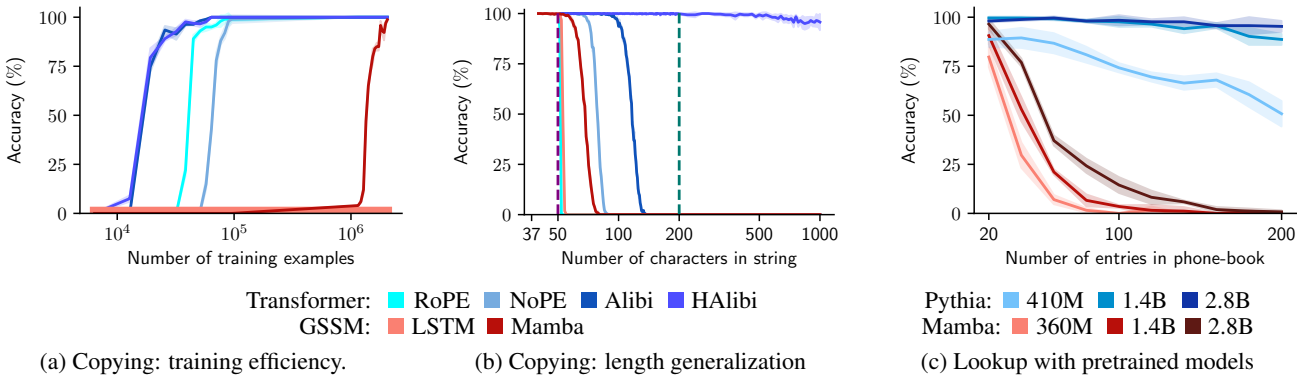
Our theory studies representation expressivity, but not whether these representations will be learned. Moreover, in practice a large GSSM may have enough capacity to represent the entire input in the latent state, at least in theory. To resolve these concerns, we conduct a variety of synthetic experiments with models of $\sim$160M parameters. We find

[1]Harvard University, Center of Mathematical Sciences and Applications [2]Harvard University, Kempner Institute for the Study of Natural and Artificial Intelligence [3]Harvard University, Departments of Computer Science and Statistics. Correspondence to: Samy Jelassi <sjelassi@fas.harvard.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

[1]In some naive implementations of transformers, it is common to allocate a $L \times L$ matrix to compute the attention. However, memory efficient implementations, such as FlashAttention (Dao et al., 2022), compute the attention with $O(L)$ memory.

[2]Note that we study copying of the input and *not* copying of training data (McCoy et al., 2023; Carlini et al., 2022)



(a) Copying: training efficiency. (b) Copying: length generalization (c) Lookup with pretrained models

Figure 1. **(a) Copying: training efficiency.** Here we train models to copy strings of length ≤ 300 and evaluate string-level accuracy on strings of length 300. Transformers train much faster than GSSMs. An LSTM cannot even learn the task within this number of samples. **(b) Copying: length generalization.** Here we train models to copy on strings of length ≤ 50 until all models are perfect in-distribution and evaluate string-level accuracy. Purple dotted line indicates maximum training string length and green dotted line indicates context window during training. Evaluating on longer inputs, the transformer models dramatically outperform the GSSMs. Using our Hard-Alibi positional encoding, we can even generalize well beyond the training context size. **(c) Lookup with pretrained models.** Here the task requires looking up and retrieving a number from a "phone book" of varying length that is entirely in context. We evaluate pretrained models 1-shot without any finetuning. Pythia (a transformer model) substantially outperforms Mamba (a GSSM) across model sizes.

that transformers are both much more efficient at learning to copy (Figure 1a) and also generalize better to longer inputs (Figure 1b). Additionally, we verify experimentally that the copy "algorithm" learned by transformers indeed relies on n-grams to perform a lookup of where to copy from (Figure 3), similarly to our theoretical construction.

Finally, we present a variety of experiments on pre-trained models to test their ability to remember and access the input context. In particular, we show that Pythia transformers (Biderman et al., 2023) outperform Mamba GSSMs (Gu & Dao, 2023) of similar size at a variety of memory-intensive tasks including copying and retrieving information from the context (Figure 1c). This is especially notable since the Mamba models achieve lower perplexity than the Pythia models at language modeling on the Pile (Gao et al., 2020). These experiments illustrate the practical relevance of the memory issues that we raise, and hint at one way that architectual choices can impact the downstream performance of LLMs above and beyond training perplexity.

2. Theory: Representational Capacity

In this section we use the copy task for a theoretical comparison between state space models and transformers. We prove two main results. First, we construct a small transformer that solves the copy task for sequences lengths that are exponential in the transformer size. Second, we show that *any* state space model *fails* to solve the copy task, unless its latent state grows linearly with the sequence length.

2.1. Setting

Let $\mathbb{D}$ be a dictionary, which contains D "alphabet" tokens. A sequence-to-sequence model is a function $H : \mathbb{D}^* \to \mathbb{D}^*$, which maps an input sequence of tokens to an output sequence. We think of the input $x_1, \ldots, x_i$ as the "prompt" to the model, and of the output sequence $H(x_1, \ldots, x_i)$ as the generated "answer".

A sequence-to-token mapping is a function $h : \mathbb{D}^* \to \mathbb{D}$. Any sequence-to-token model h naturally defines a sequence-to-sequence model H by auto-regressive inference. Namely, for every input sequence $x_1, \ldots, x_i \in \mathbb{D}$ we define recursively $x_{i+j} = h(x_1, \ldots, x_{i+j-1})$ and let $H(x_1, \ldots, x_i) = (x_{i+1}, x_{i+2}, \ldots)$.

Generalized state space models. A state space $\mathcal{S}$ is some finite set. We denote by $\mathrm{mem}(\mathcal{S})$ the number of bits required to encode the states of $\mathcal{S}$, namely $\mathrm{mem}(\mathcal{S}) = \log(|\mathcal{S}|)$. A *generalized state space model* (GSSM) is a sequence model defined by an update rule $u : \mathcal{S} \times \mathbb{D} \to \mathcal{S}$ and some output function $r : \mathcal{S} \to \mathbb{D}$. Let $s_0 \in \mathcal{S}$ be some initial state. Given some sequence $x_1, \ldots, x_L$, the state of the model at iteration i is denoted by $S_i(x_1, \ldots, x_i)$ and the output token is denoted by $R_i(x_1, \ldots, x_i)$. The state and output are defined recursively: 1) $S_0(\emptyset) = s_0$, 2) $S_i(x_1, \ldots, x_i) = u(S_{i-1}(x_1, \ldots, x_{i-1}), x_i)$, 3) $R_i(x_1, \ldots, x_i) = r(S_i(x_1, \ldots, x_i))$.

Remark 2.1. It is important to note that for any sequence model, there are two types of memory considerations: 1) input-independent memory (*parameters*) and 2) input-dependent memory (*activations*). The GSSM definition constraints the input-dependent memory (*activations*), which

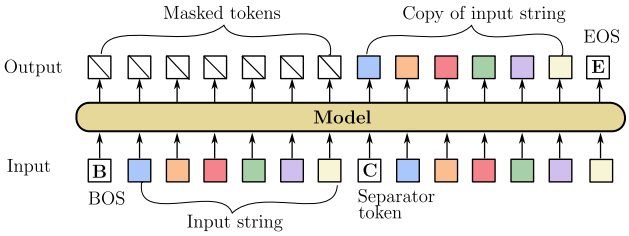


Figure 2. An illustration of the copy task.

corresponds to $\mathrm{mem}(\mathcal{S})$, and does not restrict in any way the amount of input-independent memory (*parameters*) or the run-time of state updates. Since our main goal is to show a lower bound on the state space memory, leaving all other considerations unconstrained only strengthens our results.

Transformers. Given some input of length L and dimension d, denoted $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L \in \mathbb{R}^d$, an attention head is parameterized by $W_k, W_q, W_v \in \mathbb{R}^{d \times d}$. We denote $\boldsymbol{k}_i = W_k \boldsymbol{x}_i, \boldsymbol{q}_i = W_q \boldsymbol{x}_i, \boldsymbol{v}_i = W_v \boldsymbol{x}_i$ and denote $K_i = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_i] \in \mathbb{R}^{d \times i}$ and $V_i = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_i] \in \mathbb{R}^{d \times i}$. We denote the output of the head at token i by $\boldsymbol{o}_i \in \mathbb{R}^d$, where $\boldsymbol{o}_i = V_i \cdot \mathrm{softmax}(K_i \cdot \boldsymbol{q}_i)$.

We consider a transformer with l attention heads, each one of dimension d so that the full dimension of the Transformer is dl. An embedding is some mapping $\Psi : \mathbb{D} \to \mathbb{R}^d$. An MLP is a function $f : \mathbb{R}^{dl} \to \mathbb{R}^{dl}$ s.t. $f(\boldsymbol{x}) = U_1 \sigma(U_2 \boldsymbol{x})$, for some activation function σ. Both the embedding and the MLP layer are assumed to be applied on the token level. An attention-block is a set of l heads applied in parallel, and a transformer-block is an attention-block followed by an MLP which operates on the concatenated output of the l heads. The output of the model is sampled based on the output of the final layer. For simplicity, we study the $\arg\max$ "sampling" (i.e., predicting the most probable token).

The copy task. To define the copy task, we add two special tokens to $\mathbb{D}$: (1) *beginning-of-sequence* token, denoted $\langle \textsc{bos} \rangle$, and (2) *copy* token, denoted $\langle \textsc{copy} \rangle$. So now $|\mathbb{D}| = D + 2$. A length-L copy distribution $\mathcal{D}_L$ over $\mathbb{D}^{L+2}$ generates strings of the form: "$\langle \textsc{bos} \rangle, x_1, \ldots, x_L, \langle \textsc{copy} \rangle$", where $\boldsymbol{x} \in (\mathbb{D} \setminus \{\langle \textsc{bos} \rangle, \langle \textsc{copy} \rangle\})^L$.

For some sequence-to-sequence model $H : \mathbb{D}^* \to \mathbb{D}^*$, we denote the error of H on a copy distribution $\mathcal{D}_L$ by

$$\mathrm{err}_{\mathcal{D}_L}(H) = \Pr_{\mathcal{D}_L}\left[H_{1:L}(\langle \textsc{bos} \rangle, \boldsymbol{x}, \langle \textsc{copy} \rangle) \neq \boldsymbol{x}\right]$$

where $H_{1:L}(\cdot)$ denotes the first L tokens generated by H. That is, we expect the model to output an exact copy of $\boldsymbol{x}$.

2.2. Transformers can copy inputs of exponential length

In this section, we show that transformers can implement the copy operation for input sequences with length exponential in the number of heads. Namely, we construct a transformer with two blocks that gets small error on the copy task.

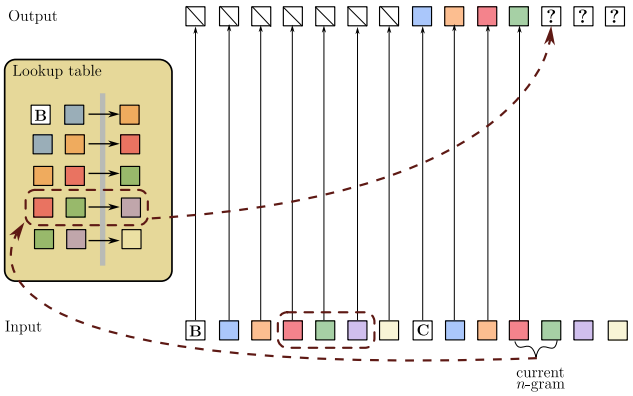


Figure 3. An illustration of the n-gram based copy algorithm. In order to predict the next token, we match the current n-gram to the corresponding n-gram in the input, then output the next token.

Construction: hash-based copying. The key idea in the construction is to first "hash" sequences of n tokens (n-grams), then at each iteration of the auto-regression attend to the previous occurrence of the most recent n-gram, and output the succeeding token. That is, we show that a transformer can implement the copying algorithm illustrated in Figure 3 (and see also Algorithm 1 in the Appendix).

Positional embedding: Hard-ALiBi. To perform the hashing described in the algorithm, we need to be able to leverage local positional information to define a hash, and also to apply this hash function globally on the entire input. To do this, we use a hard version of ALiBi (Press et al., 2021), which we call *Hard-ALiBi*. Just as in ALiBi, we add a bias b_i to the i-th attention head as follows: $\boldsymbol{o}_i = V_i \cdot \mathrm{softmax}(K_i \cdot \boldsymbol{q}_i + b_i)$. Specifically, we set b_i s.t. $b_{i,j} = -\infty$ for $j \leq i - m$ and $b_{i,j} = 0$ for $j > i - m$. We allow different heads with different choices of m and also allow for $m = \infty$ which corresponds to softmax attention with no positional embedding. This is illustrated in Figure 8c (Appendix). While the Hard-ALiBi is introduced for our theoretical construction, we observe it also offers significant benefits empirically, as discussed in Section 3.

Guarantees. The copy algorithm given in Algorithm 1 (and similarly, our transformer construction) can perfectly copy the input sequence, as long as there are no repeated n-gram patterns in the input. Therefore, the error of the algorithm depends on the probability of repeated n-grams:

Definition 2.2. Let $\mathcal{D}_L$ be some copy distribution. For some $n \in \mathbb{N}$, let $p_{\mathrm{n-gram}}(\mathcal{D}_L)$ be the probability that $x_1, \ldots, x_L$ contains two repeated sequences of n tokens. Namely:

$$p_{\mathrm{n-gram}}(\mathcal{D}_L) = \Pr_{\mathcal{D}_L}\left[\exists_{i \neq j} \text{ s.t. } x_i, \ldots x_{i+n} = x_j, \ldots, x_{j+n}\right]$$

Below we state the main theoretical result on copying with transformers, showing that transformers can copy their input, with error bounded by the probability of repeated n-grams:

Theorem 2.3. *For all n, there exists a depth-2 transformer $\mathcal{T}$ of dimension $O(n\log(D))$ s.t. for all $2n \leq L \leq D^n$, and for any copy distribution $\mathcal{D}_L$, $\mathrm{err}_{\mathcal{D}_L}(\mathcal{T}) < p_{\mathrm{n-gram}}(\mathcal{D}_L)$.*

Intuitively, the probability of repeated n-grams decays quickly when increasing the value of n. Indeed, we show that for the uniform distribution over sequences, this probability decays *exponentially* with n:

Lemma 2.4. *Let $\mathcal{D}_L$ be the copy distribution generated by sampling $\boldsymbol{x}$ from the uniform distribution over the "alphabet" (non-special) tokens. Then, $p_{\mathrm{n-gram}}(\mathcal{D}_L) < L^2 D^{-n}$.*

Combining the above results, we get that transformers can copy sequences of tokens drawn from the uniform distribution, using a number of parameters that depends only *logarithmically* on the input sequence length.

Corollary 2.5. *Fix some $\epsilon \in (0, 1/2)$ and some $L \geq \Omega(\log(1/\epsilon))$. There exists a depth-2 transformer $\mathcal{T}$ of dimension $O(\log(L/\epsilon)\log(D))$ s.t. for the uniform copy distribution $\mathcal{D}_L$, $\mathrm{err}_{\mathcal{D}_L}(\mathcal{T}) < \epsilon$.*

Remark 2.6. For simplicity we do not limit the precision of the parameters or activations, but note that our results hold for finite-precision transormers, using $O(\log(\log(L)))$ bits.

2.3. State Space Models cannot copy inputs beyond memory size

We saw that transformers are able to copy uniform sequences of tokens, with parameter count *logarithmic* in the sequence length. We now show that GSSMs *cannot* copy uniform input sequences, unless the capacity of their state space grows *linearly* with the size of the sequence length. This is intuitive: to be able to copy the entire input sequence, the model needs to store it in its state space, which requires the memory to grow linearly with the sequence length.

Theorem 2.7. *Fix some GSSM H over state space $\mathcal{S}$. Then, for all L, for the uniform copy distribution $\mathcal{D}_L$, the model H has error $\mathrm{err}_{\mathcal{D}_L}(H) > 1 - \frac{|\mathcal{S}|}{D^L}$.*

Given Theorem 2.7, the following Corollary is immediate:

Corollary 2.8. *Fix some $L \in \mathbb{N}$. Then, every GSSM H with state space $\mathcal{S}$ s.t. $\mathrm{mem}(\mathcal{S}) < L\log(D) - 1$ has error $\mathrm{err}_{\mathcal{D}_L}(H) > 1/2$ for the uniform copy distribution $\mathcal{D}_L$.*

Remark 2.9. As mentioned previously, the input-dependent memory of transformers grows linearly with the sequence length, which is *less* memory-efficient compared to GSSMs. However, it is interesting to note that from the above result, at least for the copy task, transformers are *almost optimal* in terms of their input-dependent memory. More specifically, an implication of Theorem 2.3 is that there exists a transformer which can copy inputs of length L using $\tilde{O}(L)$ input-dependent memory[3], and due to Corollary 2.8 this is indeed optimal (up to logarithmic factors).

3. Learning to Copy

In the previous section, we proved that transformers can represent the copy operation for exponentially long sequences, while GSSMs fail to copy long sequences due to their limited memory. While these results show that in theory, transformers can outperform GSSMs, our theoretical results do not establish that such a gap will be observed in practice for two reasons. First, it is not clear that transformers can indeed *learn* to copy from examples. Second, GSSMs in practice may use a large latent state memory, so that our bounds only hold for very long sequences of tokens. For example, a latent state of 1000 32-bit floating point numbers has enough bits to store at least 2000 tokens from a 50K token vocabulary. However, even though a GSSM could fit the context into memory, it may not learn to do so.

Our goal in this section is to verify that our theoretical analysis bears out experimentally when training models from scratch on synthetic data, before moving on to study pretrained models in the next section. Specifically, we train transformers and GSSMs (LSTM (Hochreiter & Schmidhuber, 1997) and Mamba (Gu & Dao, 2023)) on variants of the copy task shown in Figure 2.

3.1. Experimental setup

We now provide a brief overview of our experimental setup. Further details may be found in Appendix A.

Architecture. In all our experiments, we set the model hyperparameters so that the Mamba and transformers have a similar number of parameters (≈ 160 million parameters). Since we find that large LSTMs are hard to train (as confirmed in Pascanu et al. (2013)), we use the largest LSTM we managed to train which has ≈ 40 million parameters.

Dataset. During training, we generate in an online manner a batch of 64 examples at each epoch. At test time, we evaluate our models on 10 batches of 128 examples. We report the mean and standard-deviation over these 10 batches. If not specified otherwise, our token space $\mathcal{V}$ is of size 30 and made of the alphabet letters i.e. $\mathcal{V} = \{a, \ldots, z, \langle \textsc{bos} \rangle, \langle \textsc{eos} \rangle, \langle \textsc{copy} \rangle\}$ where $\langle \textsc{bos} \rangle$ is the beginning of sentence token, $\langle \textsc{eos} \rangle$ the end of sentence token and $\langle \textsc{copy} \rangle$ the separator token. All the strings are sampled uniformly i.e. we first sample the length of the sequence and then independently sample each position of the string from $\mathcal{V}$. Finally, we "pack the context" with i.i.d. sequences

[3] We use $\tilde{O}$ to hide logarithmic factors.

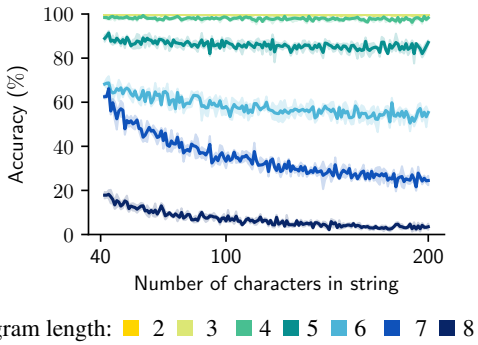


Figure 4. String-level copying accuracy on data with duplicated n-grams. Copying fails when the duplicated n-gram is too long as the model can no longer perform n-gram lookups.

during training similarly to (Zhou et al., 2023): we fill the context with multiple independent samples of the task.

Positional information. Positional information also plays an important role in the length generalization capacity of Transformers (Jelassi et al., 2023; Kazemnejad et al., 2023; Shen et al., 2023). Previously popular methods of input-layer positional embeddings (e.g. sinusoidal (Vaswani et al., 2017) or learned (Radford et al., 2019)) have been replaced by relative positional encodings at each attention layer (e.g. RoPE (Su et al., 2023), Alibi (Press et al., 2021), or NoPE (Kazemnejad et al., 2023)). Below, we experiment these positional encodings along with the Hard-Alibi encoding introduced in Section 2.

3.2. Data efficiency on the copy task

We begin by training our models on the simple task of copying a sequence of input tokens described in Figure 2. The model gets an input of $\leq L$ tokens followed by a *Separator* ($\langle\text{COPY}\rangle$) token, and needs to output the same sequence again from the beginning. In this section, we focus on in-distribution learning: we train on strings of random length $\leq L = 300$ and record the string-level accuracy on evaluation strings sampled from the training distribution. Results for this experiment are shown in Figure 1a. Clearly, there is a large gap between the transformers and GSSMs. We observe that the transformers need 100x less samples than the best GSSMs to learn the copy task.

Note that the sharp changes in accuracy displayed in Figure 1a are due to the log-scaled x-axis and choice of string-level accuracy as a metric. In Figure 9a, we report the character-level accuracy, which yields smoother curves demonstrating the learning process of GSSMs. Regarding LSTMs, we find that they do not manage to learn on length-300 strings even at the character level. In Figure 9b, we show that LSTMs are able to learn to copy on shorter strings and that string length is the bottleneck.

3.3. Length generalization on the copy task

The prior experiment demonstrates superior efficiency of learning in-distribution. Now, we test the ability of the learned functions to generalize out-of-distribution. Specifically, we consider generalization from short sequences to longer sequences. Testing this sort of generalization can help us to better understand which function the model has learned, i.e. whether the model has truly learned the "correct" copy operation or whether it just learned to copy sequences of the particular size it was trained on.

Here, we train all models on sequences of ≤ 50 tokens, and test them on sequences of up to 1000 tokens, reporting string-level accuracy. As seen in Figure 1b, all models are able to (eventually) solve the task in-distribution on lengths of ≤ 50, but transformer-based models display much better generalization to longer inputs compared to GSSMs. Namely, we observe that the performance of the GSSMs (LSTM and MAMBA) drops to zero almost immediately when increasing the input length, while the performance of transformers decays much more gradually with length.

Positional information. When looking at the relative performance of different transformer models in Figure 1b, it becomes clear that the positional encoding is important to length generalization. Specifically, the ALiBi and NoPE transformers dramatically outperform the RoPE model on longer inputs. This is likely because the sinusoidal embeddings of RoPE create a more dramatic change than the decay of ALiBi or NoPE when we go to longer inputs.

Improved generalization with Hard-ALiBi. To test our understanding of how transformers learn to copy, we now consider swapping in the Hard-ALiBi positional encoding that we used in our theoretical construction of hash-based copying (introduces in Subsection 2.2 and illustrated in Figure 8 in the Appendix). Figure 1b shows that a transformer trained with Hard-ALiBi embedding on sequences of length ≤ 50 achieves almost perfect length generalization up to sequences of length 1000. Note that this is well beyond the context length ever encountered in training.

3.4. Transformers learn to use n-gram hashing

Next, we attempt to determine whether the transformer trained on the copy task indeed applies the mechanism of storage and retrieval of n-grams. To do this, we evaluate the performance of a transformer with Hard-ALiBi positional encoding trained on the copy task when tested on a distribution of examples that intentionally contains duplicate n-grams. That is, we draw uniform sequences of tokens, and then randomly replace some n-gram with another n-gram that already appears in the sequence, such that each example always contains two copies of the same n-gram

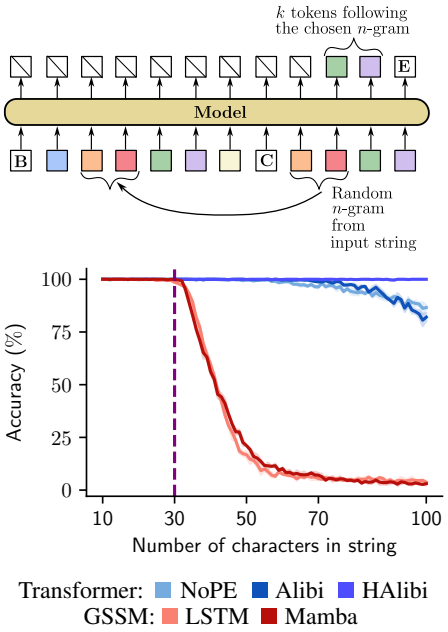


Figure 5. **Top**: An illustration of the suffix key veriant of the n-gram lookup task. **Bottom:** When trained on strings of length $\leq$ 30, transformers outperform GSSMs on longer inputs, illustrating superior performance on this memory-intensive task.

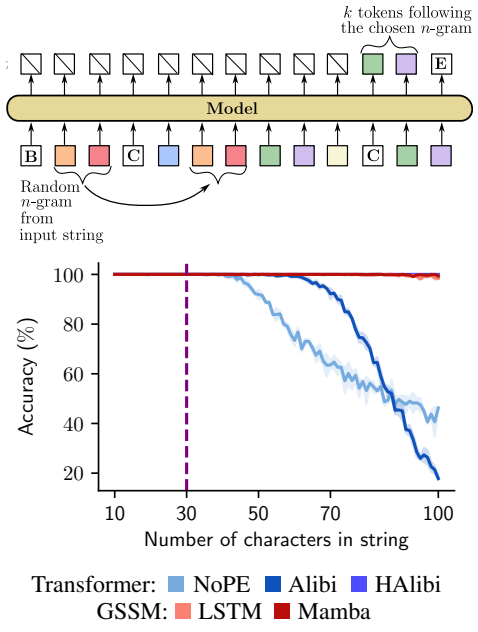


Figure 6. **Top**: An illustration of the prefix key veriant of the n-gram lookup task. **Bottom:** When trained on strings of length $\leq$ 30, GSSMs perform as well as the Hard-Alibi transformer and better than the other transformers. This slight variant of the task requires much less memory and is thus more suitable to the strengths of GSSMs at storing a small state over time.

(typically followed by a different token). We use the Hard-Alibi model here since it performs the best for the copy task as showed in Figure 1a. Figure 4 shows the performance of the transformer for different choices of n. We observe that the transformer maintains roughly the same accuracy for $n \leq 4$, but that its accuracy starts dropping when the inputs contains duplicate sequences of 5 or more tokens. This suggests that the transformer relies on something like 5-gram retrieval to do the copy task. Figure 11 further strengthens this point. We report the performance of perfect n-gram models in the copy task and observe that the performance of Transformers enhanced with Hard-ALiBi matches with the one of a 5-gram model.

3.5. GSSMs cannot arbitrarily retrieve from context

We now introduce another task to probe the mechanisms that the models use to copy from the context: the *n-gram lookup* task. In this task the model needs to use a given n-gram as a key to look up the k-token key that follows the query. We consider two variants of the task: *suffix keys* and *prefix keys*. In both variants, we assess length generalization to understand the function that the models have learned.

First, we consider the suffix key version of n-gram lookup. In this task, the model is given a sequence L of input tokens, a separator, and then an n-gram from the input sequence. The model then needs to output a sequence of k tokens following the chosen n-gram (see Figure 5 for an illustration). This task is closely related to induction heads (Olsson et al., 2022). This task requires the model to be able to "store" the entire context in order to effectively find the correct key to access it's query. We train all models on sequences of at most 30 tokens and show results in Figure 5. Transformers perform well on this task, with a relatively small drop in performance when increasing the sequence length up to 100. This suggests that transformers can learn to perform n-gram storage and retrieval. GSSMs, however, perform poorly beyond their training distribution. Intuitively, this task still requires the models to store the entire input sequence, something that GSSMs struggle to do.

Next, we try the prefix key version of n-gram lookup. Here we provide the n-gram key at the beginning and then the full input sequence (illustrated in Figure 6). In this version of the task the model does not need to store the entire input since it can look for the key on the fly as the sequence is processed. This is good for the GSSMs, since they can write the key into the state and then ignore inputs that do not match. Indeed, GSSMs achieve perfect length-generalization on this variant. Interestingly, the GSSMs even outperform the NoPE and ALiBi transformers (although not the Hard-Alibi model). We hypothesize that this may be an issue where these positional embeddings make it more difficult to effectively perform the hashing lookup over a long distance in relative positions. Taken together, these results illustrate how GSSMs seem to be memory limited, but can be effective when the tasks only require a summary of the inputs

rather than storing the entire context.

4. Pre-trained Models

In this section, we compare the performance of pre-trained transformers and pre-trained GSSMs on memory-intensive tasks such as copying long strings, retrieval and few-shot question answering. We show that transformers outperform GSSMs of similar scale on such memory-intensive tasks, even when the GSSM has lower perplexity as a language model. These results confirm that the limitation of GSSMs raised in previous sections apply to large scale models trained on real pretraining data.

4.1. Setup

In the experiments below, we compare Pythia transformer models (Biderman et al., 2023) of sizes ranging from 410M to 2.8B against Mamba models (Gu & Dao, 2023) of similar sizes. All these models have been pre-trained on the Pile (Gao et al., 2020) and use the same tokenizer. The Mamba models generally have slightly lower perplexity on the training set for a given size. The main difference between the Pythia and the Mamba models is their architectural design.

We compare these models by measuring their performance while varying the input instance length and consider two types of tasks: copy-based and information retrieval tasks. The copy-based tasks consist of presenting a random text to the model and asking it to copy the text. In the information retrieval tasks, we provide a text to the model and ask it a related question. These retrieval tasks can be seen as "selective copy", since the model needs to copy a small chunk of the input text in order to respond to the question. To measure performance, we use the string-level accuracy in all the experiments except in Figure 7c where we consider question answering and thus report the F1 score. We evaluate the models over 10 batches of size 64 for all the tasks except for question answering where we evaluate over 50 questions because the number of questions with a given context length is limited. Further details are in Appendix A.

4.2. Copying the input context

We first observe that pre-trained transformers outperform pre-trained GSSMs at copying long natural language strings. In Figure 7a, we randomly sample strings from the C4 dataset (Raffel et al., 2020) with varying number of tokens. Our prompt consists of two copies of the sampled string plus the first word of the string and we expect the model to complete the third copy. Even the smallest transformer model dramatically outperforms the largest GSSM. This happens even though the large GSSMs have enough bits in the state variable to potentially store the context. This confirms the idea that this is an architectual bias of transformers that makes it easier for them to copy from the context.

Unlike strings of tokens sampled uniformly at random, natural text can often be compressed, possibly allowing language models to copy longer strings even with limited memory. To test whether this matters, in Figure 7b we conduct the same experiment as above but randomly shuffle the order of the words in the strings. We find that when we shuffle the words, both GSSMs and transformers perform worse on the task, but the effect is more stark for GSSMs. Even the largest GSSM now gets zero accuracy on strings of length 300. This suggests that when the input is more difficult to compress, the GSSM suffers due to its fixed size state.

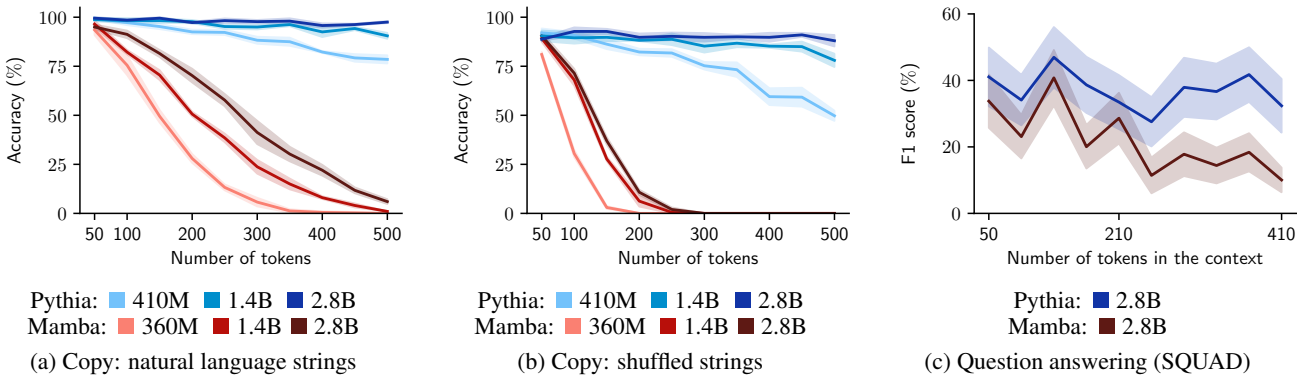
4.3. Retrieval from the input context

While copying provides a clear task to separate the model classes, it is not a particularly realistic task. That said, it presents an extreme case of a type of behavior that is highly relevant for many tasks of interest. In particular, many tasks require retrieving specific information from the context that is relevant to the desired output. This subsection presents examples of how our results transfer to more practical tasks.

Phone-book lookup. We first consider a "phone-book" experiment where we provide a synthetic phone-book to the model and ask it to return the phone number when given a name. We generate the phone-book by randomly sampling L names and their associated phone number. One line of this phone-book looks like "John Powell: 609-323-7777". Our prompt to the model consists of the phone-book, two few-shot examples and a question asking for the phone number of a randomly sampled name from the phone-book. Figure 1c reports the accuracy obtained by the pretrained transformers and GSSMs while varying the size of the phone-book L. We observe that even the smallest transformer (410M parameters) outperforms the largest GSSMs (2.8B parameters) when the phone-book size is long enough ($L \geq 70$). This shows that in retrieval tasks which require access to the whole context, GSSMs struggle to store the relevant information in their fixed-size state.

Question-Answering. In this experiment, we compare the 2.8B parameter Mamba and transformer models[4], on the SQuAD question-answering dataset (Rajpurkar et al., 2018). This dataset provides text paragraphs together with a few questions regarding the text. We probe the models to answer the question by providing a single demonstration of a question/answer pair (corresponding to the same text) before giving the target question. We bin the paragraphs according to their lengths, and report the F1 score as a function of the paragraph length for both models in Figure 7c. We

[4]In our experiments, smaller models were unable to achieve reasonable and consistent performance on this dataset.



(a) Copy: natural language strings (b) Copy: shuffled strings (c) Question answering (SQUAD)

Figure 7. **(a) Copy: natural language strings.** We compare pretrained models on their ability to copy natural language strings sampled from C4 of varying lengths and report string-level accuracy. The transformer models substantially outperform the GSSMs. **(b) Copy: shuffled strings.** To test whether it mattered that the strings were in natural language, we randomly shuffle the word order of the strings from the previous experiment. We find that this degrades performance, especially for the Mamba models. **(c) Question answering (SQUAD).** We compare Pythia and Mamba on a standard question answering dataset where we bin the dataset based on the length of the context paragraph. We find that Mamba performance decays more quickly with the length of the context.

observe that while for short paragraphs, both the Pythia transformer and Mamba achieve comparable performance, the performance of Mamba degrades more quickly with the paragraph length, while the transformer-based model maintains a similar accuracy even for longer texts. This result shows that the fixed-memory of GSSMs also limits their performance on standard natural tasks.

5. Related Work

There exists a broad body of prior work on the representational capacity of GSSMs like RNNs (Merrill, 2019; Merrill et al., 2020) as well as transformers (Weiss et al., 2021; Merrill et al., 2022; Wei et al., 2022; Sanford et al., 2023; Edelman et al., 2022). Previous works that study transformers do so through comparison to other complexity classes, such as threshold circuits (Merrill et al., 2022), RASP language (Weiss et al., 2021) or first-order logic (Chiang et al., 2023) (see Strobl et al. (2023) for a thorough review). These works do not provide insights into how transformers implement algorithms for solving specific problems. In contrast, our theoretical result constructs a transformer for the copy task, which illustrates the mechanism and provides tight bounds on the model size. Together with the result showing that GSSMs cannot copy long sequences, our theory characterizes the power of different sequence models on the copy task. Other theoretical separation results between transformers and RNNs (Sanford et al., 2023; Merrill, 2019) use more complex tasks of less practical relevance.

Other papers have previously demonstrated the capacity of transformers to leverage the entire input context for tasks like retrieval, question answering, and in-context learning (Devlin et al., 2018; Raffel et al., 2020; Petroni et al., 2020; Brown et al., 2020; Liu et al., 2023b; Kamradt, 2023). Another line of work has studied the "induction head" mechanism in transformers that performs a retrieval operation much like the one we observe for copying (Olsson et al., 2022). But, to our knowledge, there is not a comparison in related work between transformers and GSSMs of similar quality on these tasks.

Several of our experiments study length generalization as a way to assess whether the model found the "right way" to solve the task. Prior work on length generalization in transformers has focused on the data distribution (Anil et al., 2022), positional embeddings (Kazemnejad et al., 2023), and arithmetic tasks (Delétang et al., 2022; Ruoss et al., 2023; Jelassi et al., 2023; Zhou et al., 2023). We extend many of these ideas to the copying task.

Finally, we note that while we focus on tasks where transformers outperform GSSMs, there are also tasks where GSSMs outperform transformers. For example, Liu et al. (2023a) shows that transformers fail to generalize out of distribution for "flip-flop language modeling", while LSTMs do so easily. These tasks require tracking a small $O(1)$ state variable over time. Another benefit of GSSMs is the ability to input long contexts like DNA sequences that may be impractical for transformers (Nguyen et al., 2023).

Concurrently to our work, Akyürek et al. (2024); Grazzi et al. (2024); Park et al. (2024) studied the difference between Transformers and Mamba at in-context learning, which can be seen as a form of copying. In particular, Akyürek et al. (2024) finds that Transformers have an advantage over other architectures at this task because they have "n-gram heads". Similarly to these works, we hint the limitations of SSMs in memory-intensive tasks such as copying because of their limited state size. We also show that Transformers can

perform copying using the Hard-ALiBi positional encoding, which improves the model's ability to learn n-gram matching.

6. Discussion

We have demonstrated through theory and experiments that transformers are better than GSSMs at copying from their input context. However, we emphasize that state space models have many advantages over transformers. The memory and computational complexity of GSSMs does not increase with the input length, which is ideal for training and inference on long inputs. Additionally, state space models such as RNNs are better at tracking state variables across long sequences (Liu et al., 2023a), which may be useful for generating long consistent text. Importantly, language processing in the human brain appears to be much more similar to how state space models process language (Tikochinski et al., 2024).

We therefore believe that future work should focus on building hybrid architectures that endow state space models with an attention-like mechanism, allowing them to retrieve relevant pieces of text from their input. Indeed, humans have an incredibly limited capacity for memorizing sequences (Miller, 1956), but can translate entire novels if we allow them to look back at the text (Shelton, 1612).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

We thank Boaz Barak for helpful discussions. Kempner Institute computing resources enabled this work. Samy Jelassi acknowledges funding supported by the Center of Mathematical Sciences and Applications. This work has been made possible in part by a gift from the Chan Zuckerberg Initiative Foundation to establish the Kempner Institute for the Study of Natural and Artificial Intelligence. Sham Kakade acknowledges funding from the Office of Naval Research under award N00014-22-1-2377.

References

Akyürek, E., Wang, B., Kim, Y., and Andreas, J. In-context language learning: Arhitectures and algorithms. *arXiv preprint arXiv:2401.12973*, 2024.

Anil, C., Wu, Y., Andreassen, A., Lewkowycz, A., Misra, V., Ramasesh, V., Slone, A., Gur-Ari, G., Dyer, E., and Neyshabur, B. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Bradbury, J., Merity, S., Xiong, C., and Socher, R. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., and Zhang, C. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2022.

Chiang, D., Cholak, P., and Pillay, A. Tighter bounds on the expressivity of transformer encoders. *arXiv preprint arXiv:2301.10743*, 2023.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Delétang, G., Ruoss, A., Grau-Moya, J., Genewein, T., Wenliang, L. K., Catt, E., Hutter, M., Legg, S., and Ortega, P. A. Neural networks and the chomsky hierarchy. *arXiv preprint arXiv:2207.02098*, 2022.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Edelman, B. L., Goel, S., Kakade, S., and Zhang, C. Inductive biases and variable creation in self-attention mechanisms. In *International Conference on Machine Learning*, pp. 5793–5831. PMLR, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Grazzi, R., Siems, J., Schrodi, S., Brox, T., and Hutter, F. Is mamba capable of in-context learning? *arXiv preprint arXiv:2402.03170*, 2024.

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

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jelassi, S., d'Ascoli, S., Domingo-Enrich, C., Wu, Y., Li, Y., and Charton, F. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*, 2023.

Kamradt, G. Llmtest_needleinahaystack. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Kazemnejad, A., Padhi, I., Ramamurthy, K. N., Das, P., and Reddy, S. The impact of positional encoding on length generalization in transformers. *arXiv preprint arXiv:2305.19466*, 2023.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Exposing attention glitches with flip-flop language modeling. *arXiv preprint arXiv:2306.00946*, 2023a.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023b.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

McCoy, R. T., Smolensky, P., Linzen, T., Gao, J., and Celikyilmaz, A. How much do language models copy from their training data? evaluating linguistic novelty in text generation using raven. *Transactions of the Association for Computational Linguistics*, 11:652–670, 2023.

Merrill, W. Sequential neural networks as automata. *arXiv preprint arXiv:1906.01615*, 2019.

Merrill, W., Weiss, G., Goldberg, Y., Schwartz, R., Smith, N. A., and Yahav, E. A formal hierarchy of rnn architectures. *arXiv preprint arXiv:2004.08500*, 2020.

Merrill, W., Sabharwal, A., and Smith, N. A. Saturated Transformers are Constant-Depth Threshold Circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 08 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00493. URL https://doi.org/10.1162/tacl_a_00493.

Miller, G. A. The magic number seven plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63:91–97, 1956.

Nguyen, E., Poli, M., Faizi, M., Thomas, A., Birch-Sykes, C., Wornow, M., Patel, A., Rabideau, C., Massaroli, S., Bengio, Y., et al. Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *arXiv preprint arXiv:2306.15794*, 2023.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Park, J., Park, J., Xiong, Z., Lee, N., Cho, J., Oymak, S., Lee, K., and Papailiopoulos, D. Can mamba learn how to learn? a comparative study on in-context learning tasks. *arXiv preprint arXiv:2402.04248*, 2024.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. Pmlr, 2013.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Petroni, F., Lewis, P., Piktus, A., Rocktäschel, T., Wu, Y., Miller, A. H., and Riedel, S. How context affects language models' factual predictions. *arXiv preprint arXiv:2005.04611*, 2020.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners, 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Rajpurkar, P., Jia, R., and Liang, P. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Ruoss, A., Delétang, G., Genewein, T., Grau-Moya, J., Csordás, R., Bennani, M., Legg, S., and Veness, J. Randomized positional encodings boost length generalization of transformers. *arXiv preprint arXiv:2305.16843*, 2023.

Sanford, C., Hsu, D., and Telgarsky, M. Representational strengths and limitations of transformers. *arXiv preprint arXiv:2306.02896*, 2023.

Shelton, T. *The Ingenious Gentleman Don Quixote of La Mancha*. 1612. Written by Miguel de Cervantes, translated by Thomas Shelton.

Shen, R., Bubeck, S., Eldan, R., Lee, Y. T., Li, Y., and Zhang, Y. Positional description matters for transformers arithmetic. *arXiv preprint arXiv:2311.14737*, 2023.

Strobl, L., Merrill, W., Weiss, G., Chiang, D., and Angluin, D. Transformers as recognizers of formal languages: A survey on expressivity. *arXiv preprint arXiv:2311.00208*, 2023.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, pp. 127063, 2023.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Tikochinski, R., Goldstein, A., Meiri, Y., Hasson, U., and Reichart, R. An incremental large language model for long text processing in the brain. 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wei, C., Chen, Y., and Ma, T. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *Advances in Neural Information Processing Systems*, 35:12071–12083, 2022.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

A. Experimental setup

In this section, we provide additional details about our experimental setup. We first give a description of the positional encodings used in our transformers experiments (Subsection A.1) and then give details about the training and evaluation procedures (Subsection A.2).

A.1. Positional encodings in the transformers

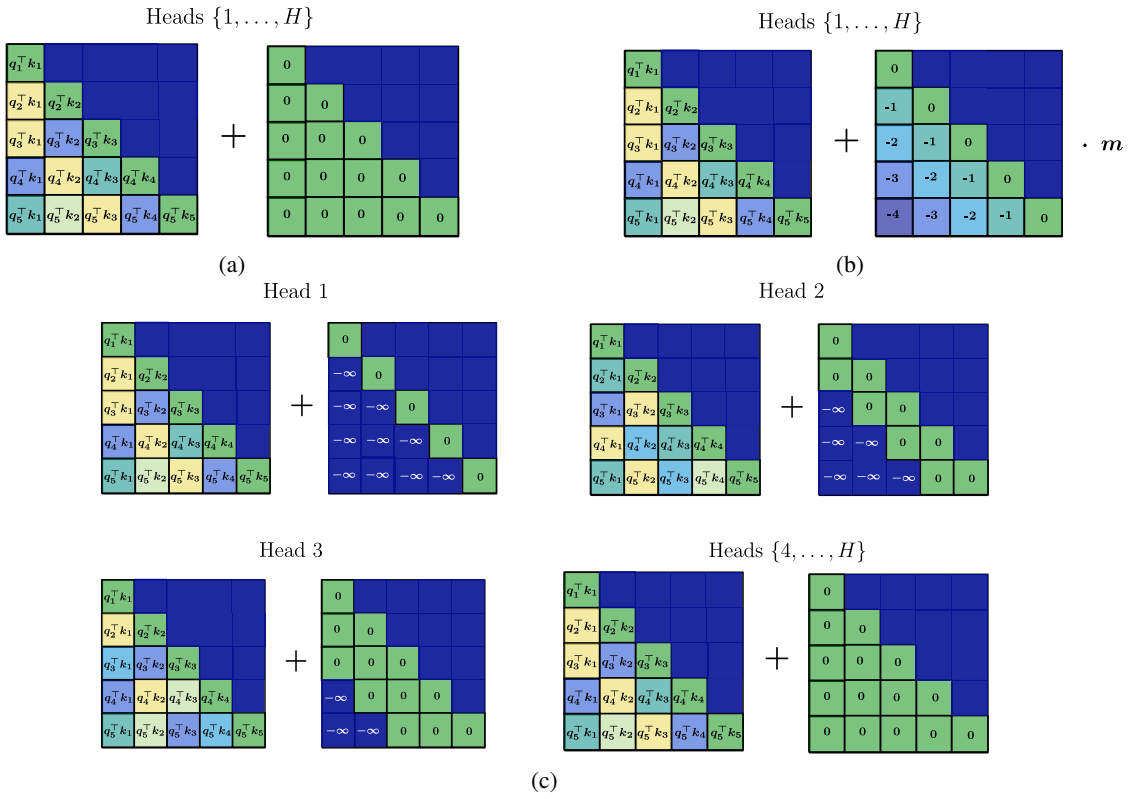


Figure 8. **Positional encoding schemes for transformers**: illustration of the different positional encodings of the transformers that are trained in our experiments. (a) corresponds to the NoPE encoding (Kazemnejad et al., 2023) where no positional encoding is applied to any of the attention heads (b) depicts the ALiBi encoding (Press et al., 2021) where m is a head-specific scalar and (c) the Hard-ALiBi encoding introduced in Section 2. For the sake of illustration, we consider the case where we mask three heads which means that we force Heads 1, 2 and 3 to attend to their current token, their current and preceding tokens and their current, preceding and prior to the preceding tokens. The remaining heads are set as NoPE heads.

We consider multiple positional encoding schemes in our experiments in Section 3:

– the NoPE scheme (Kazemnejad et al., 2023) where no positional information is added to any of the attention scores (Figure 8a). This architecture choice helps to get better length generalization in multiple tasks including the copy task.

– the ALiBi scheme (Press et al., 2021) which biases the attention scores with a penalty that is proportional to their distance (Figure 8b). m is a head-specific slope fixed before training.

– the Hard-ALiBi scheme introduced in Section 2 which has M masked attention heads where we explicitly force the model to attend to their directly previous tokens and $H - M$ heads set to be NoPE attention heads. In Figure 8c, we display the case where we have $M = 4$ masked heads: in the first head, the tokens just attend to themselves; in the second head, the tokens attend to themselves and to previous ones; in the third head, the tokens attend to themselves, the previous ones and the second preceding tokens. The remaining $H - M$ heads are set to NoPE.

A.2. Pretraining and evaluation details

Software dependencies. We implement all of our training in Pytorch (Paszke et al., 2019). We use the HuggingFace library (Wolf et al., 2019) and the Mamba GitHub repository (Gu & Dao, 2023).

Architectures. In our experiments in Section 3, the backbone of our transformers is the GPT-NeoX architecture. We set the number of layers to 12, the hidden size to 1024 and the number of heads $H = 16$. We consider the different positional encodings that are described in Subsection A.1. For Alibi, we set the head-specific scalar as in the original paper i.e. $m_h = 2^{-h/2}$ for $h \in \{1, \dots, H\}$. For the Hard-Alibi model, we sweep over the number of masked heads $M \in \{2, \dots, 10\}$ and found that the best model corresponds to $M = 6$. Regarding the Mamba models, we set the number of layers to 24 and the hidden size 1024. We also sweep over the state space dimension $S \in \{16, 32, 64, 128, 256\}$ and found the best model is $S = 32$. This choice of hyperparameters ensures that both transformers and Mamba models have a comparable number of parameters. Lastly, our LSTM is made of 4 layers and width 1024.

Training hyperparameters. In Section 3, at each epoch, we sample online a batch size of size 64. We fill the context with examples so we choose a context length ($C = 420$ for all the experiments except Figure 1a where we set $C = 620$) and pack as many examples as possible to fit this context. So in our case, one sample contains many instances. We run the experiments for 15 epochs for both transformers and Mamba while for LSTMs we need 300 epochs. All methods are trained with the AdamW optimizer (Loshchilov & Hutter, 2017) with learning rate 5e-5, a linear rate decay schedule, 300 steps of warmup and default weight decay of 1e-1. Finally, to train all the models, we use the next-token prediction loss but we apply a mask on the input instance so that we only penalize the model whenever it makes a mistake on the labels (and not on the inputs and labels jointly).

Compute resources. Pretraining was all done on an internal cluster using RTX8000 GPUs. We estimate that the final training run needed to produce the results in the paper took approximately 600 GPU hours.

Evaluation algorithm. We evaluate the models over 10 batches of size 64 for all the tasks except for the question answering one where we evaluate over 50 questions because the number of questions with a given context length is limited.

Decoding algorithm. At inference, all our models use greedy decoding for generation and we set the temperature to 0.

B. Additional Experiments

In Subsection B.1, we focus on the in-distribution learning of the copy task and show that the number of samples needed by GSSMs is much higher than the one for transformers. In Subsection B.2, we study the performance of pre-trained models on the copy task in the case where the strings are sampled uniformly. This experiment shows that when the text to copy is totally random, the gap between pre-trained transformers and GSSMs is even larger.

B.1. Data efficiency on the copy task

In this section, we provide additional plots to complement the data efficiency experiment from Figure 1a. We want to highlight the following points:

- in Figure 1a, we see a sharp transition for the Mamba learning curve. However, Figure 9a shows that the learning process is more smooth at the character level. Besides, LSTMs are not able to learn the copy on length-300 strings even at the character level.
- We consider the experiment of learning to copy much shorter strings namely strings with length ≤ 30. Figure 9b shows that the gap in terms of training examples between transformers and Mamba is much smaller i.e. Mamba only needs 10x more data. Besides, we see that the LSTM is able to learn the copy task but it needs 100x more data than transformers.

B.2. Pre-trained models on the uniform copy task

In this section, we provide an additional experiment that shows the superiority of pre-trained Pythia over pre-trained Mamba models in the copy task.

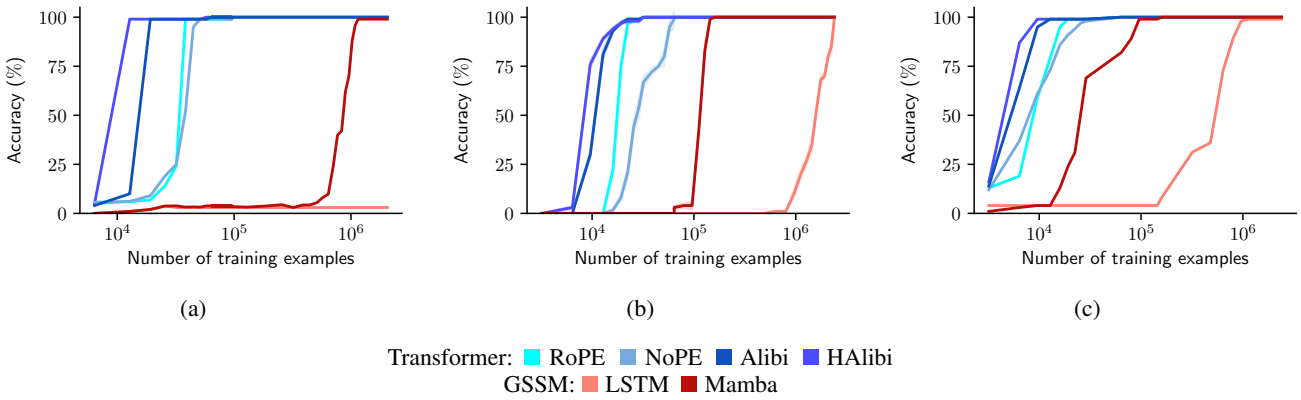


Figure 9. **(a) Copying long strings: character-level accuracy.** Here we train models to copy strings of length ≤ 300 and evaluate character-level accuracy on strings of length 300. Transformers train much faster than GSSMs. Mamba has a more progressive learning curve than in Figure 1a. An LSTM cannot even learn the task within this number of samples at the character level. **(b) Copying short strings: string-level accuracy.** Here we train models to copy strings of length ≤ 30 and evaluate character-level accuracy on strings of length 30. Transformers train much faster than GSSMs. Compared to Figure 1a, we see that Mamba needs way less samples in order to learn to copy length-30 strings. An LSTM can learn to copy but requires 100x more training examples. **(c) Copying short strings: character-level accuracy.** Here we train models to copy strings of length ≤ 30 and evaluate character-level accuracy on strings of length 30 and report the character-level accuracy.

We consider the same setup as in Section 3: we sample uniform strings of alphabet characters with a fixed length and ask the model to copy it by using the same prompt format as the one described in Subsection 4.2.

This setting is a more extreme version of Figure 7b since the strings are more random: in Figure 7b, the order of the nouns were random but the nouns were English nouns while in Figure 7b, the strings are totally random. In Figure 10, we see a clear separation between the transformers and Mamba models with the smallest Pythia outperforming the largest Mamba. However, compared to Figure 7b, the Pythia performance is much higher since the 1.4B model able to get almost 100% accuracy.

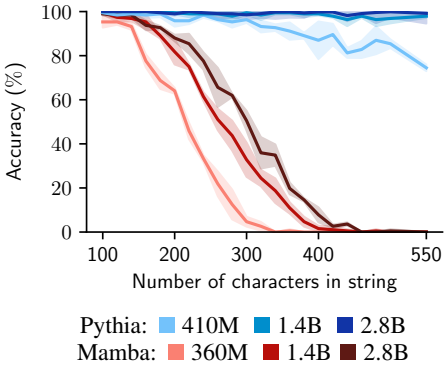


Figure 10. **Copy: uniform strings.** To test whether it mattered that the strings were in natural language, we generate uniformly sampled strings (the generation process is described in Section 3). We find that this degrades the Mamba models while Pythia models are able to keep a high performance.

B.3. Performance n-gram models at copying

In Figure 11, we display the performance of perfect n-gram models in the copy task. To obtain these curves, we uniformly sample 128 strings over 3 seeds and report the probability there is a n-gram. This probability corresponds to the performance of a perfect n-gram model. We observe that Transformers enhanced with the Hard-ALiBi positional encoding have a performance close to a perfect 5-gram model.

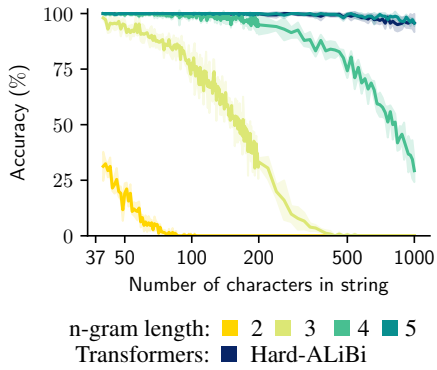


Figure 11. String-level copying accuracy obtained by perfect n-gram models and Transformers with Hard-ALiBi. Transformers performance matches the one of 5-gram model.

Algorithm 1 Hash-based copying

Input: sequence $x_1, \ldots, x_L$
Let $s : \mathbb{D}^* \to \mathbb{R}^d$ be some hashing function.
for $i = n+2, \ldots, L$ **do**
  $k_i \leftarrow s(x_{i-n}, x_{i-n+1}, \ldots, x_{i-1})$
  $v_i \leftarrow x_i$
end for
for $j = 1, \ldots, L$ **do**
  if $j \leq n$ **then**
    $y_j \leftarrow x_j$
  else
    $q_j \leftarrow s(y_{j-n}, \ldots, y_{j-1})$
    Let $i \in [L]$ s.t. $k_i = q_j$, and set $y_j \leftarrow x_i$
  end if
end for
Output: sequence $y_1, \ldots, y_L$

C. Proofs - Upper Bound

This section gives a detailed proof of Theorem 2.3 and Lemma 2.4.

C.1. Technical Lemmas

We begin by introducing some technical lemmas that we use in the proof of Theorem 2.3.

Lemma C.1. *Let $h_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = \frac{1}{\min(t,i)} \sum_{j=\max(1,i-t+1)}^{i} \boldsymbol{x}_j$. Then, h_t can be computed using a hard-ALiBi attention head.*

Proof. Let $W_k, W_q = 0$ (zero matrix) and let $W_v = I_d$ (indentity matrix). We choose $b_i \in \{0, -\infty\}^i$ s.t.

$$b_{i,j} = \begin{cases} -\infty & j \leq i - t \\ 0 & j > i - t \end{cases}$$

□

Lemma C.2. *Assume that $d = \lceil \log(D) \rceil + 2$. Then, there exists an embedding Ψ s.t.*

- *For every $x \in \mathbb{D}$ it holds that $\|\Psi(x)\|_2 = 1$ and $\|\Psi(x)\|_\infty \leq 1$.*
- *For $x' \neq x$ it holds that $\langle x, x' \rangle < 1 - \frac{1}{d}$.*
- *For every $x \neq \langle \text{BOS} \rangle$, $\langle \Psi(x), \Psi(\langle \text{BOS} \rangle) \rangle = 0$, and for every $x \neq \langle \text{COPY} \rangle$, $\langle \Psi(x), \Psi(\langle \text{COPY} \rangle) \rangle = 0$.*

Proof. Denote $d' = \lceil \log(D) \rceil$, and observe that we can encode all D "non-special" tokens as vectors in $\left\{ \pm \frac{1}{\sqrt{d}} \right\}^{d'}$, and denote this encoding by Ψ'. Now, define:

$$\Psi(x) = \begin{cases} [1, 0, \ldots, 0] & x = \langle \text{BOS} \rangle \\ [0, 1, 0, \ldots, 0] & x = \langle \text{COPY} \rangle \\ [0, 0, \Psi'(x)] & o.w. \end{cases}$$

□

Lemma C.3. *Let $\boldsymbol{z} \in \mathbb{R}^K$ be some vector such that, for some constants $a > b > 0$, there exists $i \in [K]$ s.t. $z_i = a$ and for all $j \neq i$ we have $|z_j| \leq b$. Denote $\boldsymbol{s} = \mathrm{softmax}(\boldsymbol{z})$. Then $s_i \geq \frac{1}{1+K\exp(b-a)}$ and $s_j \leq \exp(b-a)$ for all $j \neq i$.*

Proof. First, notice that:

$$\exp(a) = \exp(z_i) \leq \sum_{j=1}^{K} \exp(z_j) \leq \exp(z_i) + (K-1)\exp(b) \leq \exp(a) + K\exp(b) = \exp(a)(1 + K\exp(b-a))$$

Observe the following:

$$s_i = \frac{\exp(z_i)}{\sum_{j=1}^{K}\exp(z_j)} \geq \frac{\exp(a)}{\exp(a)(1+K\exp(b-a))} = \frac{1}{1+K\exp(b-a)}$$

Finally, for every $j \neq i$:

$$s_j = \frac{\exp(z_j)}{\sum_{j=1}^{K}\exp(z_j)} \leq \frac{\exp(b)}{\exp(a)} = \exp(b-a)$$

□

C.2. Proof of Theorem 2.3

We begin by constructing the first block of the transformer, which computes the "lookup-table" for the copy algorithm. This lookup-table consists of pairs of (key,values) for each position i, where the key encodes the n-gram preceding the i-th token, and the value is the i-th token. Namely, if the sequence is $x_1, \ldots, x_i$, then $\mathrm{key}_i = (x_{i-n-1}, \ldots, x_i)$ and $\mathrm{value}_i = x_i$. Additionally, the transformer block also computes a query, which is just the "current" n-gram, i.e. $\mathrm{query}_i = (x_{i-n}, \ldots, x_i)$. The copy algorithm matches the current query with previous key-s, retrieving the matching value.

The following theorem shows that by using a combination of n hard-ALiBi attention heads (with different choice of m for each head), together with an MLP layer, can compute the correct $(\mathrm{key}_i, \mathrm{value}_i, \mathrm{query}_i)$ for each position. We use a slightly modified $\mathrm{key}_i, \mathrm{query}_i$ to handle cases where the $i \leq n$ (or, i is one of the first n tokens after the $\langle \textsc{copy} \rangle$ token).

Lemma C.4. *Let Ψ be the one-hot embedding. Then, there exists a hard-ALiBi transformer block with 3 outputs, denoted $T^{\mathrm{key}}, T^{\mathrm{query}}, T^{\mathrm{value}}$, which correspond to 3 blocks of the output dimension, s.t. $T^{\mathrm{key}} : \mathbb{R}^{d\times *} \to \mathbb{R}^{(d+1)n\times *}$, $T^{\mathrm{query}} : \mathbb{R}^{d\times *} \to \mathbb{R}^{(d+1)n\times *}$ and $T^{\mathrm{value}} : \mathbb{R}^{d\times *} \to \mathbb{R}^{d\times *}$ satisfying, for all $\boldsymbol{x}$ sampled from a length-n copy distribution,*

1. *Value output: for all i,*
$$T_i^{\mathrm{value}}(\Psi(x_1), \ldots, \Psi(x_i)) = \Psi(x_i)$$
2. *Key output:*
   - *For $t = 1, \ldots, n$, if $i > n$*
   $$T^{\mathrm{key}}_{(t-1)d+1:td,i}(\Psi(x_1), \ldots, \Psi(x_i)) = \Psi(x_{i-t})$$
   and if $i \leq n$
   $$T^{\mathrm{key}}_{(t-1)d+1:td,i}(\Psi(x_1), \ldots, \Psi(x_i)) = 0$$
   - *Additionally, for $t = 1, \ldots, n$, for all i*
   $$T^{\mathrm{key}}_{nd+t,i}(\Psi(x_1), \ldots, \Psi(x_i)) = \mathbf{1}\{i = t+1\}$$
3. *Query output:*
   - *For $t = 1, \ldots, n$, if $i \geq n$*
   $$T^{\mathrm{query}}_{(t-1)d+1:td,i}(\Psi(x_1), \ldots, \Psi(x_i)) = \Psi(x_{i-t+1})$$
   and if $i < n$
   $$T^{\mathrm{query}}_{(t-1)d+1:td,i}(\Psi(x_1), \ldots, \Psi(x_i)) = 0$$

- *Additionally, for $t = 1, \ldots, n$, for all i*

$$T^{\text{key}}_{nd+t,i}(\Psi(x_1), \ldots, \Psi(x_i)) = n \cdot \mathbf{1}\{i = L + t\}$$

Proof. We prove the following:

1. For the value output, we simply take $T^{\text{value}} = h_1$ as defined in Lemma C.1.
2. For each $t = 0, \ldots, n$, define:

$$g_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = (t+1) \cdot h_{t+1}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) - t \cdot h_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$$

where we define $h_0 \equiv 0$. Observe that if $i > t$ then:

$$g_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = (t+1) \cdot \frac{1}{t+1} \sum_{j=i-t}^{i} \boldsymbol{x}_j - t \cdot \frac{1}{t} \sum_{j=i-t+1}^{i} \boldsymbol{x}_j = \boldsymbol{x}_{i-t}$$

and if $i \leq t$ then:

$$g_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = t \cdot \frac{1}{i} \sum_{j=1}^{i} \boldsymbol{x}_j - (t-1) \cdot \frac{1}{i} \sum_{j=1}^{i} \boldsymbol{x}_j = \frac{1}{i} \sum_{j=1}^{i} \boldsymbol{x}_j$$

For every $j \in [d]$, denote

$$\begin{aligned}\hat{g}_{t,j}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = \sigma(e_j \cdot g_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) - n\Psi(\langle \text{BOS} \rangle) \cdot g_n(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)) \\ - \sigma(-e_j \cdot g_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) - n\Psi(\langle \text{BOS} \rangle) \cdot g_n(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i))\end{aligned}$$

Claim: $\hat{g}_t(\Psi(x_1), \ldots, \Psi(x_i)) = \mathbf{1}\{i > n\} \cdot \Psi(x_{i-t})$

Proof: Fix some $j \in [d]$. Observe that for all i, $|e_j \cdot g_t(\Psi(x_1), \ldots, \Psi(x_i))| \leq 1$.

- If $i \leq n$, we have $g_n(\Psi(x_1), \ldots, \Psi(x_i)) = \frac{1}{i} \sum_{j'=1}^{i} \Psi(x_{j'})$ and so $\Psi(\langle \text{BOS} \rangle) \cdot g_n(\Psi(x_1), \ldots, \Psi(x_i)) = 1$ where we use the properties of Ψ and the fact that $x_1 = \langle \text{BOS} \rangle$. Therefore, $\hat{g}_{t,j}(\Psi(x_1), \ldots, \Psi(x_i)) = 0$.
- If $i > n \geq t$, then:

$$\begin{aligned}\hat{g}_{t,j}(\Psi(x_1), \ldots, \Psi(x_i)) &= \sigma\left(e_j \cdot \Psi(x_{i-t}) - n\Psi(\langle \text{BOS} \rangle) \cdot \Psi(x_{i-t})\right) \\ &\quad - \sigma\left(-e_j \cdot \Psi(x_{i-t}) - n\Psi(\langle \text{BOS} \rangle) \cdot \Psi(x_{i-t})\right) \\ &= \sigma(e_j \cdot \Psi(x_{i-t})) - \sigma(-e_j \cdot \Psi(x_{i-t})) = e_j \cdot \Psi(x_{i-t})\end{aligned}$$

  where we use the fact that $x_{i-t} \neq \langle \text{BOS} \rangle$ and therefore $\Psi(\langle \text{BOS} \rangle) \cdot \Psi(x_{i-t}) = 0$.

Denote

$$\tilde{g}_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = \frac{1}{2}\sigma\left(2\Psi(\langle \text{BOS} \rangle) \cdot (g_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) - h_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)) - 1\right)$$

Claim: $\tilde{g}_t(\Psi(x_1), \ldots, \Psi(x_i)) = \mathbf{1}\{i = t+1\}$

Proof: Denote $g_{t,i} = g_t(\Psi(x_1), \ldots, \Psi(x_i))$ and $h_{1,i} = h_1(\Psi(x_1), \ldots, \Psi(x_i))$. Observe:

- If $i = t+1$, then $g_{t,i} = \Psi(x_1) = \Psi(\langle \text{BOS} \rangle)$ and $h_{1,i} = \Psi(x_i) \perp \Psi(\langle \text{BOS} \rangle)$ and therefore $\tilde{g}_{t,i} = 1$.
- If $i > t+1$ then $g_{t,i} = \Psi(x_{i-t}) \perp \Psi(\langle \text{BOS} \rangle)$ and $h_{1,i} = \Psi(x_i) \perp \Psi(\langle \text{BOS} \rangle)$ and so $\tilde{g}_{t,i} = 0$.
- If $1 < i \leq t$ then $\Psi(\langle \text{BOS} \rangle) \cdot g_{t,i} = \frac{1}{i} \leq \frac{1}{2}$ and $h_{1,i} = \Psi(x_i) \perp \Psi(\langle \text{BOS} \rangle)$ and so $\tilde{g}_{t,i} = 0$.
- If $i = 1$ then $g_{t,i} = h_{1,i} = \Psi(\langle \text{BOS} \rangle)$ and therefore $\tilde{g}_{t,i} = 0$.

Finally, we can take $T^{\text{key}} = [\hat{g}_1, \ldots, \hat{g}_q, \tilde{g}_1, \ldots, \tilde{g}_q]$.

3. For all $t = 1, \ldots, n$, define $g^*_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = \sigma(\Psi(\langle \text{COPY} \rangle) \cdot g_{t-1}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i))$.

Claim: $g^*_t(\Psi(x_1), \ldots, \Psi(x_i)) = \mathbf{1}\{i = L + t\}$

Proof: Denote $g_{t,i} = g_t(\Psi(x_1), \ldots, \Psi(x_i))$. Observe:

- If $i = L + t$ then $g_{t-1,i} = \Psi(x_{i-t+1}) = \Psi(x_{L+1}) = \Psi(\langle \text{COPY} \rangle)$ and therefore $g^*_{t,i} = 1$.
- If $i \neq L + t$ and $i > t - 1$ then $g_{t-1,i} = \Psi(x_{i-t+1}) \perp \Psi(\langle \text{COPY} \rangle)$ and therefore $g^*_{t,i} = 0$.
- If $i \leq t$ then since $x_1, \ldots, x_i \neq \langle \text{COPY} \rangle$ we get $\Psi(\langle \text{COPY} \rangle) \cdot g_{t-1,i} = 0$ and therefore $g^*_{t,i} = 0$.

Therefore, we can take $T^{\text{query}} = [\hat{g}_0, \ldots, \hat{g}_{q-1}, n \cdot g^*_1, \ldots, n \cdot g^*_q]$.

□

Now, we prove Theorem 2.3 by showing that using a single attention head with no positional embedding on top of the construction in Lemma C.4 realizes the copy algorithm. Since the first block computes the correct choice of $\text{key}_i, \text{query}_i, \text{value}_i$, by correctly scaling of the attention matrix we verify that the output of the second layer at position i corresponds to $\approx \text{value}_j$ for j s.t. $\text{key}_j = \text{query}_i$.

Proof of Theorem 2.3. Let $T^{\text{value}}, T^{\text{key}}, T^{\text{query}}$ be the outputs of the Transformer block guaranteed by Lemma C.4. Observe that, for some temprature $\tau \in \mathbb{R}$, the following function can be computed by a softmax-attention layer on-top of this block:

$$H(\Psi(x_1), \ldots, \Psi(x_i)) = T^{\text{value}} \cdot \text{softmax}(\tau \cdot T^{\text{key}} \cdot T^{\text{query}}_i)$$

where e.g. T^{value} denotes $T^{\text{value}}(\Psi(x_1), \ldots, \Psi(x_i))$.

For now, assume that all the n-grams in $\boldsymbol{x}$ are unique, and that the length of the input satisfies $2L + 2 \leq K$ for $K = D^n$.

Claim: Fix some $i > L$, denote $\boldsymbol{z} = T^{\text{key}} \cdot T^{\text{query}}_i$. Then, $z_{i-L+1} = n$ and $|z_j| < n - \frac{1}{d}$ for all $j \neq i - L + 1$.

Proof: We separate to the following cases:

- If $i > L + n - 1$, then for every j we have

$$\begin{aligned} T^{\text{key}}_j \cdot T^{\text{query}}_i &= \mathbf{1}\{j > n\} \cdot [\Psi(x_{j-1}), \ldots, \Psi(x_{j-n})]^\top [\Psi(x_i), \ldots, \Psi(x_{i-n+1})] \\ &= \mathbf{1}\{j > n\} \cdot \sum_{t=1}^{n} \Psi(x_{j-t}) \Psi(x_{i-t+1}) \end{aligned}$$

  Now, if $j = i - L + 1$ then $x_{j-t} = x_{i-L+1-t} = x_{i-t+1}$ and since $j > n$ we get

$$T^{\text{key}}_j \cdot T^{\text{query}}_i = \sum_{t=1}^{n} \|\Psi(x_{i-t+1})\| = n$$

  If $j \neq i - L + 1$, since there are no repeated n-grams, there is at least some $t \in [n]$ s.t. $\Psi(x_{j-t}) \neq \Psi(x_{i-t+1})$ and by the choice of the embedding $\Psi(x_{j-t}) \cdot \Psi(x_{i-t+1}) \leq 1 - \frac{1}{d}$. In this case, we get $\left| T^{\text{key}}_j \cdot T^{\text{query}}_i \right| \leq n - \frac{1}{d}$.

- If $L < i \leq L + n - 1$ and $j \leq n$ then

$$T^{\text{key}}_j \cdot T^{\text{query}}_i = n e_{j-1} \cdot e_{i-L} = n \cdot \mathbf{1}\{j = i - L + 1\}$$

  which satisfies the required.

- If $L < i \leq L + n - 1$ and $j > n$ then

$$T^{\text{key}}_j \cdot T^{\text{query}}_i = \sum_{t=1}^{n} \Psi(x_{j-t}) \Psi(x_{i-t+1})$$

  and as before, since there are no repeated n-grams, we get $\left| T^{\text{key}}_j \cdot T^{\text{query}}_i \right| \leq n - \frac{1}{d}$

Claim: Fix some $\epsilon \in (0, 1)$ and some $i > L$, denote $\boldsymbol{s} = \text{softmax}(\tau T^{\text{key}} \cdot T^{\text{query}}_i) = \text{softmax}(\tau \cdot \boldsymbol{z})$. If $\tau = d \ln(\frac{2K}{\epsilon})$, then $s_{i-L+1} \geq 1 - \epsilon$ and $s_j \leq \frac{\epsilon}{2K}$ for all $j \neq i - L + 1$.

Proof: Using the previous claim, together with Lemma C.3, we get that:

- $s_{i-L+1} \geq \frac{1}{1+i\exp(-\tau/d)} \geq \frac{1}{1+K\exp(-\tau/d)} \geq \frac{1}{1+\epsilon/2} = 1 - \frac{\epsilon/2}{1+\epsilon/2} \geq 1-\epsilon$
- For $j \neq i-L+1$,

$$s_j \leq \exp(-\tau/d) \leq \frac{\epsilon}{2K}$$

Claim: Fix some $\epsilon \in (0,1)$ and some $i > L$. Then, for $\tau \geq d\ln(\frac{2K}{\epsilon})$, it holds that:

$$\|H(\Psi(x_1),\ldots,\Psi(x_i)) - \Psi(x_{i-L+1})\| \leq \epsilon$$

Proof: Let $\boldsymbol{s}$ as defined in the previous claim. Then:

$$\begin{aligned}\|H(\Psi(x_1),\ldots,\Psi(x_i)) - \Psi(x_{i-L+1})\| &= \left\|\sum_{j=1}^{i} s_j\Psi(x_j) - \Psi(x_{i-L+1})\right\| \\ &\leq (1-s_{i-L+1})\left\|\Psi(x_{i-L+1})\right\| + \sum_{j\neq i-L+1} s_j\left\|\Psi(x_j)\right\| \\ &= (1-s_{i-L+1}) + \sum_{j\neq i-L+1} s_j \leq \epsilon + (i-1)\frac{\epsilon}{2K} \leq 2\epsilon\end{aligned}$$

Now, denote by $\Phi : \mathbb{R}^d \to \mathbb{D}$ the output map given by $\Phi(\boldsymbol{z}) = \arg\max_{x\in\mathbb{D}} \boldsymbol{z}\cdot\Psi(x)$ (which can be computed by an $\arg\max$ over a linear function).

Claim: If $\tau \geq d\ln(8Kd)$, then for all $i > L$ we have $\Phi(H(\Psi(x_1),\ldots,\Psi(x_i))) = x_{i-L+1}$.

Proof: Denote $\boldsymbol{y}_i = H(\Psi(x_1),\ldots,\Psi(x_i))$. First, using the previous claim, we observe that

$$\begin{aligned}\boldsymbol{y}_i\cdot\Psi(x_{i-L+1}) &= (\boldsymbol{y}_i - \Psi(x_{i-L+1}))\cdot\Psi(x_{i-L+1}) + \|\Psi(x_{i-L+1})\| \\ &\geq 1 - \|\boldsymbol{y}_i - \Psi(x_{i-L+1})\| \geq 1 - \frac{1}{4d}\end{aligned}$$

Next, observe that for all $j \neq i-L+1$ we have

$$\begin{aligned}\boldsymbol{y}_i\cdot\Psi(x_j) &= (\boldsymbol{y}_i - \Psi(x_{i-L+1}))\cdot\Psi(x_j) + \Psi(x_j)\cdot\Psi(x_{i-L+1}) \\ &\leq \|\boldsymbol{y}_i - \Psi(x_{i-L+1})\| + 1 - \frac{1}{d} \leq 1 - \frac{3}{4d} < \boldsymbol{y}_i\cdot\Psi(x_{i-L+1})\end{aligned}$$

From the above claim, the Transformer construction outputs the correct token at each step of the auto-regressive generation. □

C.3. Proof of Lemma 2.4

Proof of Lemma 2.4. Fix some $i < j \in [L]$. Let $I := \{i,\ldots,i+n\}$ and $J := \{j,\ldots,j+n\}$. We first bound the probability of drawing some $\boldsymbol{x}$ s.t. $\boldsymbol{x}_I = \boldsymbol{x}_J$. Note that there are $D^{|I\cup J|}$ choices for $\boldsymbol{x}_{I\cup J}$. We count the number of choices for $\boldsymbol{x}_{I\cup J}$ s.t. $\boldsymbol{x}_I = \boldsymbol{x}_J$. Notice that in this case, $\boldsymbol{x}_{I\cup J}$ is determined by $\boldsymbol{x}_{I\setminus J}$, therefore there are $D^{|I\setminus J|}$ possible choices. We conclude that

$$\Pr\left[\boldsymbol{x}_I = \boldsymbol{x}_J\right] = \frac{D^{|I\setminus J|}}{D^{|I\cup J|}} = D^{|I\setminus J|-|I\cup J|} = D^{-n}$$

Using the union bound, we get that

$$\Pr\left[\exists i < j \text{ s.t. } \boldsymbol{x}_{i,\ldots,i+n} = \boldsymbol{x}_{j,\ldots,j+n}\right] \leq \sum_{i<j}\Pr\left[\boldsymbol{x}_{i,\ldots,i+n} = \boldsymbol{x}_{j,\ldots,j+n}\right] < L^2D^{-n}$$

□

D. Proofs - Lower Bound

In this section, we prove Theorem 2.7. We begin by showing that, for every input, the output of the model in each iteration is a deterministic function of the state of the model after observing the input:

Lemma D.1. *Let $H_{u,r} : \mathbb{D}^{n'} \to \mathbb{D}^n$ be some fixed-state sequence-to-sequence model. Then, there exists map $G : \mathcal{S} \to \mathbb{D}^n$ s.t. for all $\boldsymbol{x} \in \mathbb{D}^{n'}$*

$$H_{u,r}(\boldsymbol{x}) = G \circ S_{n'}(\boldsymbol{x})$$

Proof. Let $x_{n'+1}, \ldots, x_{n'+n}$ be the outputs of $H_{u,r}$. We need to show that there exist functions $G_1, \ldots, G_n$ s.t. $H_{u,r}(x_1, \ldots, x_{n'}) = G(S_{n'}(x_1, \ldots, x_n))$. We give the following recursive definition:

- $G_1(s) = r(s), \tilde{G}_1(s) = u(s, G_1(s))$.
- $G_i(s) = r(\tilde{G}_{i-1}(s)), \tilde{G}_i(s) = u(\tilde{G}_{i-1}(s), G_i(s))$.

Denote $s = S_{n'}(x_1, \ldots, x_{n'})$ We prove by induction that $G_i(s) = x_{n'+i}$ and also that $\tilde{G}_i(s) = S_{n'+i}(x_1, \ldots, x_{n'+i})$.

- $G_1(s) = r(s) = R_{n'}(x_1, \ldots, x_{n'}) = x_{n'+1}$.
- $\tilde{G}_1(s) = u(s, G_1(s)) = u(s, x_{n'+1}) = S_{n'+1}(x_1, \ldots, x_{n'+1})$
- $G_i(s) = r(\tilde{G}_{i-1}(s)) = r(S_{n'+i-1}(x_1, \ldots, x_{n'+i-1})) = R_{n'+i-1}(x_1, \ldots, x_{n'+i-1}) = x_{n'+i}$
- $\tilde{G}_i(s) = u(\tilde{G}_{i-1}(s, G_i(s))) = u(S_{n'+i-1}(x_1, \ldots, x_{n'+i-1}), x_{n'+i}) = S_{n'+i}(x_1, \ldots, x_{n'+i})$

and so the required follows. □

Given the previous Lemma, we bound the error of the model by comparing the number of possible states to the number of possible inputs.

Proof of Theorem 2.7. From Lemma D.1, there exists some function $G : \mathcal{S} \to \mathbb{D}^n$ s.t. $H_{u,r} = G \circ S_{n'}$. For each $\boldsymbol{x}$, we denote by $\tilde{\boldsymbol{x}}$ the sequence $\langle \textsc{bos} \rangle, \boldsymbol{x}, \langle \textsc{copy} \rangle$. Now, observe the following:

$$\begin{aligned}
1 - \mathrm{err}_{\mathcal{D}_n}(H_{u,r}) &= \Pr_{\mathcal{D}_n}[H_{u,r}(\tilde{\boldsymbol{x}}) = \boldsymbol{x}] \\
&= \frac{1}{D^n} \sum_{\boldsymbol{x} \in \mathbb{D}^n} \mathbf{1}\{H_{u,r}(\tilde{\boldsymbol{x}}) = \boldsymbol{x}\} \\
&= \frac{1}{D^n} \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{x} \in S_{n+2}^{-1}(\tilde{\boldsymbol{x}})} \mathbf{1}\{G \circ S_{n'+2}(\tilde{\boldsymbol{x}}) = \boldsymbol{x}\} \\
&= \frac{1}{D^n} \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{x} \in S_{n+2}^{-1}(\tilde{\boldsymbol{x}})} \mathbf{1}\{G(s) = \boldsymbol{x}\} \leq \frac{|\mathcal{S}|}{D^n}
\end{aligned}$$

□