FORGETTING TRANSFORMER: SOFTMAX ATTENTION WITH A FORGET GATE

Anonymous authors
Paper under double-blind review

ABSTRACT

An essential component of modern recurrent sequence models is the *forget gate*. While Transformers do not have an explicit recurrent form, we show that a forget gate can be naturally incorporated into Transformers by down-weighting the unnormalized attention scores in a data-dependent way. We name the resulting model the *Forgetting Transformer*. We show that the Forgetting Transformer outperforms the Transformer on long-context language modeling, length extrapolation, and short-context downstream tasks, while performing on par with the Transformer on long-context downstream tasks. Moreover, it is fully compatible with the Flash Attention algorithm and does not require any positional embeddings. Several analyses, including the needle-in-the-haystack test, show that the Forgetting Transformer also retains the Transformer's superior long-context capabilities over recurrent sequence models such as Mamba-2, HGRN2, and DeltaNet. We also introduce a "Pro" block design that incorporates some common architectural components in recurrent sequence models and find it significantly improves the performance of both the Forgetting Transformer and the Transformer.

1 INTRODUCTION

Despite the growing interest in reviving recurrent sequence models (Gu et al., 2021; Peng et al., 2021; Yang et al., 2023; Gu & Dao, 2023; Sun et al., 2023; De et al., 2024; Qin et al., 2024b; Dao & Gu, 2024; Peng et al., 2024; Beck et al., 2024; Zhang et al., 2024), these models still underperform the Transformer (Vaswani et al., 2017) in terms of *long-context capabilities* (Hsieh et al., 2024; Waleffe et al., 2024; Shen et al., 2024; Qin et al., 2024a), likely due to their relatively small fixed-sized hidden states (Jelassi et al., 2024). While the Transformer excels in handling long-context information, it lacks an explicit mechanism for forgetting past information in a *data-dependent* way. Such a mechanism – often implemented as some form of the *forget gate* (Gers et al., 2000) – is ubiquitous in recurrent sequence models and has proven critical in their success in short-context tasks (Greff et al., 2016; Van Der Westhuizen & Lasenby, 2018; Peng et al., 2021; Yang et al., 2023; Gu & Dao, 2023). A natural question to ask is then: can we have a forget gate in Transformers?

To address this question, we leverage an important fact: many recurrent sequence models with a forget gate can be written in a parallel linear attention form (Katharopoulos et al., 2020) analogous to softmax attention (Yang et al., 2023; Dao & Gu, 2024). In this parallel form, the forget gate mechanism translates into down-weighing the unnormalized attention scores in a data-dependent way. Our key insight is that this *exact* mechanism is also applicable to softmax attention. We name the resulting model the *Forgetting Transformer*.

We show that the Forgetting Transformer outperforms the Transformer on long-context language modeling, length extrapolation, and short-context downstream tasks, while performing on par with the Transformer on long-context downstream tasks. Notably, it does not require any positional embeddings. It also retains the ability of the Transformer to perform accurate long-context retrieval and achieves perfect accuracy in a simplified needle-in-the-haystack test (Kamradt, 2023). In contrast, all the tested recurrent sequence models fail. We also introduce a "Pro" block design that integrates several architectural components commonly used in recurrent sequence models, which significantly improves the performance of both the Forgetting Transformer and the baseline Transformer. Finally, we show that the Forgetting Transformer can be implemented in a hardware-aware way with a simple modification to the Flash Attention (Dao, 2023) algorithm.

2 BACKGROUND: LINEAR ATTENTION WITH A FORGET GATE

This section introduces the notation used in this work and gives a brief background on linear attention. We also introduce a gated variant of linear attention and discuss its parallel form, which naturally leads to the Forgetting Transformer. Throughout this work, we only consider causal sequence modeling. We also mainly consider the single-head case; extension to the multi-head case is straightforward.

2.1 LINEAR ATTENTION

Standard causal softmax attention takes a sequence of input vectors $(\boldsymbol{x}_i)_{i=1}^L$ and produces a sequence of output vectors $(\boldsymbol{o}_i)_{i=1}^L$, where $\boldsymbol{x}_i, \boldsymbol{o}_i \in \mathbb{R}^d, i \in \{1, \dots, L\}$. Each $\boldsymbol{o}_i$ is computed as follows:

$$\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i = \boldsymbol{W}_q \boldsymbol{x}_i, \boldsymbol{W}_k \boldsymbol{x}_i, \boldsymbol{W}_v \boldsymbol{x}_i \in \mathbb{R}^d, \tag{1}$$

$$\boldsymbol{o}_i = \frac{\sum_{j=1}^i k_{\exp}(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^i k_{\exp}(\boldsymbol{q}_i, \boldsymbol{k}_j)} = \frac{\sum_{j=1}^i \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^i \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j)}, \tag{2}$$

where $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{d \times d}$ are projection matrices and $k_{\exp}(\boldsymbol{q}, \boldsymbol{k}) = \exp(\boldsymbol{q}^\top \boldsymbol{k})$ is the exponential dot product kernel.[1]

Linear attention (Katharopoulos et al., 2020) replaces the exponential dot product kernel $k_{\exp}(\boldsymbol{q}, \boldsymbol{k}) = \exp(\boldsymbol{q}^\top \boldsymbol{k})$ with a kernel $k_\phi(\boldsymbol{q}, \boldsymbol{k})$ with some feature representation $\phi : \mathbb{R}^d \to (\mathbb{R}^+)^{d'}$:

$$\boldsymbol{o}_i = \frac{\sum_{j=1}^i k_\phi(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^i k_\phi(\boldsymbol{q}_i, \boldsymbol{k}_j)} = \frac{\sum_{j=1}^i (\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j))\boldsymbol{v}_j}{\sum_{j=1}^i \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)} \tag{3}$$

Following Yang et al. (2023), we call the above the *parallel form* of linear attention as it can be computed with matrix multiplications. Alternatively, linear attention can be computed in a *recurrent form*:

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{v}_t \phi(\boldsymbol{k}_t)^\top \tag{4}$$

$$\boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \phi(\boldsymbol{k}_t) \tag{5}$$

$$\boldsymbol{o}_t = \frac{\boldsymbol{S}_t \phi(\boldsymbol{q}_t)}{\boldsymbol{z}_t^\top \phi(\boldsymbol{q}_t)}, \tag{6}$$

where $\boldsymbol{S}_t \in \mathbb{R}^{d \times d'}, \boldsymbol{z}_t \in \mathbb{R}^{d'}, t \in \{0, \dots, L\}$ are computed recurrently, with $\boldsymbol{S}_0 = \boldsymbol{0}$ and $\boldsymbol{z}_t = \boldsymbol{0}$.

2.2 LINEAR ATTENTION WITH A FORGET GATE

The recurrent form of linear attention makes it natural to introduce a forget gate. Specifically, we can compute a scalar forget gate $f_t = \sigma(\boldsymbol{w}_f^\top \boldsymbol{x}_t + b_f) \in \mathbb{R}$ at each timestep, where σ is the sigmoid function and $\boldsymbol{w}_f \in \mathbb{R}^d, b_f \in \mathbb{R}$ are learnable parameters. The recurrent computation is then:

$$\boldsymbol{S}_t = f_t \boldsymbol{S}_{t-1} + \boldsymbol{v}_t \phi(\boldsymbol{k}_t)^\top \tag{7}$$

$$\boldsymbol{z}_t = f_t \boldsymbol{z}_{t-1} + \phi(\boldsymbol{k}_t) \tag{8}$$

$$\boldsymbol{o}_t = \frac{\boldsymbol{S}_t \phi(\boldsymbol{q}_t)}{\boldsymbol{z}_t^\top \phi(\boldsymbol{q}_t)}. \tag{9}$$

Note that this gated variant of linear attention differs from most models in the literature. In particular, most gated variants of linear attention models such as GLA (Yang et al., 2023) and Mamba-2 (Dao & Gu, 2024) do not have the normalization term (i.e., there is no $\boldsymbol{z}_t$ and the output is just $\boldsymbol{o}_t = \boldsymbol{S}_t \phi(\boldsymbol{q}_t)$). We keep the normalization term to maintain similarity with softmax attention. The most similar model is gated-RFA (Peng et al., 2021), with the only difference being the lack of a $(1 - f_t)$ term in the recurrence.

[1] Note we omit the $\frac{1}{\sqrt{d}}$ scaling factor to reduce visual clutter. In practice we always scale $\boldsymbol{q}_i^\top \boldsymbol{k}_j$ by $\frac{1}{\sqrt{d}}$.

Crucially, similar to the normalization-free version derived in GLA and Mamba-2, we can show that this gated variant of linear attention also has a parallel form:

$$\boldsymbol{o}_i = \frac{\sum_{j=1}^{i} F_{ij}\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^{i} F_{ij}\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)} = \frac{\sum_{j=1}^{i} F_{ij}k_\phi(\boldsymbol{q}_i,\boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^{i} F_{ij}k_\phi(\boldsymbol{q}_i,\boldsymbol{k}_j)}, \tag{10}$$

where $F_{ij} = \prod_{l=j+1}^{i} f_l$. We adopt the convention that $F_{ij} = 1$ if $i = j$.

Our key observation is that Equation 10 and the softmax attention in Equation 2 are very similar in form. In fact, if we just change the kernel k_ϕ in Equation 10 back to the exponential dot product kernel $k_{\exp}$, we obtain the *softmax attention with a forget gate*. We introduce this formally in the next section.

3 Forgetting Transformer

Our proposed model, the *Forgetting Transformer* (also abbreviated as *FoT* throughout this work), features a modified softmax attention mechanism with a forget gate. We name this attention mechanism the *Forgetting Attention*. Similar to the gated variant of linear attention introduced in the previous section, we first compute a scalar forget gate $f_t = \sigma(\boldsymbol{w}_f^\top \boldsymbol{x}_t + b_f) \in \mathbb{R}$ for each timestep t. The output of the attention is then

$$\boldsymbol{o}_i = \frac{\sum_{j=1}^{i} F_{ij}\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^{i} F_{ij}\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j)} = \frac{\sum_{j=1}^{i} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j + d_{ij})\boldsymbol{v}_j}{\sum_{j=1}^{i} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j + d_{ij})}, \tag{11}$$

where $F_{ij} = \prod_{l=j+1}^{i} f_l$ and $d_{ij} = \sum_{l=j+1}^{i} \log f_l$. This can be written in matrix form:

$$\boldsymbol{D} = \log \boldsymbol{F} \in \mathbb{R}^{L\times L}, \tag{12}$$

$$\boldsymbol{O} = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top + \boldsymbol{D})\boldsymbol{V} \in \mathbb{R}^{L\times d}, \tag{13}$$

where $\boldsymbol{F} \in \mathbb{R}^{L\times L}$ is a lower triangular matrix whose non-zero entries are F_{ij}, i.e., $\boldsymbol{F}_{ij} = F_{ij}$ if $i \geq j$ and 0 otherwise. We adopt the convention that $\log 0 = -\infty$. $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{O} \in \mathbb{R}^{L\times d}$ are matrices containing $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i, \boldsymbol{o}_i, i \in \{1, \ldots, L\}$ as the rows. The $\mathrm{softmax}$ operation is applied row-wise. For multi-head attention with h heads, we maintain h instances of forget gate parameters $(\boldsymbol{w}_f^{(i)})_{i=1}^h$ and $(b_f^{(i)})_{i=1}^h$ and compute the forget gates separately for each head.

Hardware-aware implementation The logit bias form on the rightmost side of Equation 11 allows the Forgetting Attention to be computed with a simple modification to the Flash Attention (Dao, 2023) algorithm. Here we briefly describe the forward pass. The backward pass follows a similar idea. First, we compute the cumulative sum $c_i = \sum_{l=1}^{i} \log f_l$ for $i \in \{1, \ldots, L\}$ and store it in HBM. Note that this allows us to compute $d_{ij} = c_i - c_j$ easily later. Whenever we compute the attention logit via the dot product $\boldsymbol{q}_i^\top \boldsymbol{k}_j$ in the GPU's fast shared memory (SRAM) (as in Flash Attention), we also load c_i and c_j to SRAM, compute d_{ij}, and add the bias to the attention logit. The rest of the forward pass remains the same as Flash Attention. This algorithm avoids instantiating $L \times L$ d_{ij} matrices on the slow high-bandwidth memory of the GPU. We provide a detailed algorithm description in Appendix D. Moreover, since the forget gates are scalars instead of vectors, the additional computation and parameter count introduced are negligible.

Connection to ALiBi Besides its natural connection to gated linear attention, the Forgetting Attention can also be seen as a data-dependent and learnable version of ALiBi (Press et al., 2021). ALiBi applies a data-independent bias $b_{ij} = -(i-j)m_h$ to the attention logits, where m_h is a fixed slope specific to each head h. It is easy to show that ALiBi is equivalent to Forgetting Attention with a fixed, head-specific, and data-independent forget gate $f_t = \exp(-m_h)$. In Section 4.5, we verify the superiority of Forgetting Attention over ALiBi-based attention.

Positional embeddings Though we find that using Rotary Position Embeddings (RoPE) (Su et al., 2024) slightly improves the performance of the Forgetting Transformer, it is not necessary as it is for the standard Transformer. For simplicity, *we do not use RoPE or any other positional embeddings for the Forgetting Transformer by default*. This is discussed further in Section 4.5.

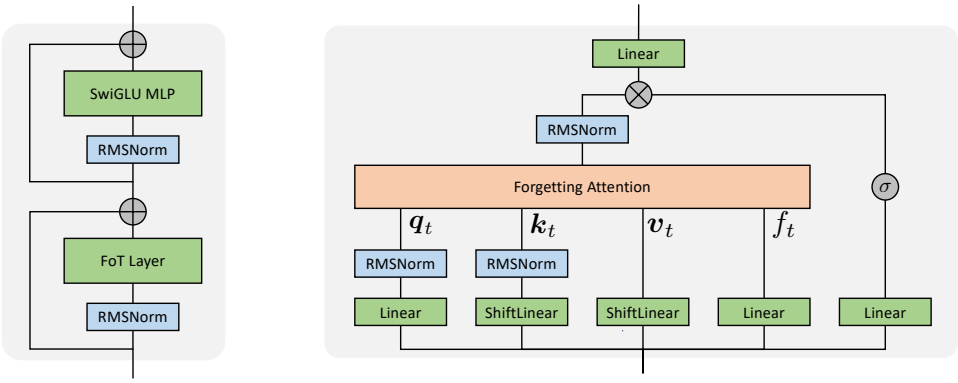

Figure 1: Default architecture of FoT. (**left**) A single FoT block. (**right**) A single FoT (Pro) time-mixing layer. All RMSNorms on the right are applied independently to each head. σ is the sigmoid function. $\otimes$ represents element-wise multiplication. The `ShiftLinear` layer implements the computation in Equation 14.

Architecture design Forgetting Attention can be used as a drop-in replacement for standard softmax attention in any Transformer architecture. In this work we test two different architectures. First, we replace softmax attention + RoPE in the LLaMA architecture (Touvron et al., 2023) with Forgetting Attention and refer to this model as *FoT (LLaMA)*. Second, we test an improved "Pro" architecture that incorporates output gates[2] and output normalization used in GLA and Mamba-2 (with slight variations). We also use QK-norm (Dehghani et al., 2023) and apply a simplified variant of data-dependent token shift (Peng et al., 2024) to the keys and values (KV-shift). Concretely, to compute the keys $(\boldsymbol{k}_i)_{i=1}^L$, we perform the following computation with additional parameters $\boldsymbol{w}_k \in \mathbb{R}^d$ for each timestep t:

$$\begin{aligned} \tilde{\boldsymbol{k}}_t &= \boldsymbol{W}_k \boldsymbol{x}_t \in \mathbb{R}^d, \quad \alpha_t^{\text{key}} = \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_t) \in \mathbb{R} \\ \boldsymbol{k}_t &= \mathrm{RMSNorm}(\alpha_t^{\text{key}} \tilde{\boldsymbol{k}}_{t-1} + (1 - \alpha_t^{\text{key}}) \tilde{\boldsymbol{k}}_t) \end{aligned} \tag{14}$$

The values $(\boldsymbol{v}_i)_{i=1}^L$ are computed in the same way, but without RMSNorm. As discussed in Olsson et al. (2022), token shift may be beneficial for forming induction heads. The overall architecture is shown in Figure 1 and detailed in Appendix A. We refer to the resulting model as *FoT (Pro)*.

4 Empirical Study

The advantage of Transformers in long-context capabilities over recurrent sequence models have been demonstrated multiple times (Hsieh et al., 2024; Waleffe et al., 2024; Shen et al., 2024; Qin et al., 2024a). However, a forget gate introduces a *recency bias*. It is thus natural to ask whether the Forgetting Transformer still maintains this advantage. Therefore, our empirical study places a special focus on long-context capabilities.

4.1 Experimental Setup

Dataset We focus on long-context language modeling and train all models on LongCrawl64 (Buckman, 2024). LongCrawl64 is a filtered long-sequence subset of RedPajama-v2 (Together Computer, 2023). It consists of pre-tokenized sequences truncated to exactly 65536 tokens.The sequences are tokenized with the TikToken tokenizer (OpenAI, 2022) for GPT-2 (Radford et al., 2019).

Baselines We are interested in two types of comparisons. First, to understand the benefits of the forget gate, we compare our proposed models with the standard Transformer. For the standard Transformer, we also test both the LLaMA architecture and our enhanced Pro architecture (referred

[2]When output gates are used, we reduce the number of parameters in the MLPs so the total number of parameters remains the same.

to as *Transformer (LLaMA)* and *Transformer (Pro)* repsectively). Similar to Xiong et al. (2023), we find it crucial to use a large RoPE angle θ for the Transformer. Following Xiong et al. (2023) we use $\theta = 500000$.

Second, to show the advantage of the Forgetting Transformer over recurrent sequence models in long-context capabilities, we compare with Mamba-2 (Dao & Gu, 2024), HGRN2 (Qin et al., 2024a), and DeltaNet (Yang et al., 2024). Notably, all of them have reported better language modeling perplexity and *short-context* downstream task performance than Transformers. The implementation of all models is based on the Flash Linear Attention repository (Yang & Zhang, 2024).

Training setup Due to limited compute resources, for our main experiments, we train models with 760M non-embedding parameters on a 15×2^{30}-token (roughly 16B tokens) subset of LongCrawl64 with a training context length of 16384 tokens. This roughly matches the compute-optimal model size/data ratio in Chinchilla scaling law (Hoffmann et al., 2022). For the validation set, we use a 2×2^{30}-token (roughly 2.1B tokens) subset of the LongCrawl64 held-out set consisting of sequences of 65536 tokens. We choose a much longer validation context length than the training context length to test the length generalization capabilities of the models.

All models are trained with AdamW (Loshchilov, 2017) with $(\beta_1, \beta_2) = (0.9, 0.95)$. We use a linear learning rate warmup from 0 to 1.25×10^{-3} for the first 256×2^{20} tokens and then a cosine decay schedule to 1.25×10^{-4}. Each training batch contains 0.5×2^{20} (roughly 0.5M) tokens. All models use a weight decay of 0.1 and gradient clipping of 1.0. We use `bfloat16` mixed-precision training for all models. More details of the experimental setup can be found in Appendix B.

4.2 Long-context language modeling

Metrics Before we present our results, it is important to understand one of our main metrics: *per-token* loss on the validation set at different token positions. To be precise, let V be the vocabulary size, $\boldsymbol{y}_i^{(j)} \in \{0,1\}^V$ be the one-hot vector encoding the language modeling target for the i-th token in the j-th validation sequence, and $\boldsymbol{p}_i^{(j)} \in \mathbb{R}^V$ be the corresponding output probabilities of the model, then the per-token loss $L(i)$ at token position i is simply

$$L(i) = \frac{1}{M} \sum_{j=1}^{M} -\log[(\boldsymbol{p}_i^{(j)})^\top \boldsymbol{y}_i^{(j)}], \tag{15}$$

where M is the number of validation sequences.

The per-token loss is particularly meaningful for understanding the long-context capabilities of a model. Informally, a monotonically decreasing $L(i)$ with a steep slope indicates the model is using the full context well. On the other hand, if $L(i)$ plateaus after some token position k, it indicates the model struggles to use tokens that are k tokens away from the current token position for its prediction. This correspondence between the slope of $L(i)$ and the model's context utilization is explained in more detail in Appendix C.

Besides per-token loss, we also report perplexity over different validation context lengths $P(l)$. Specifically, perplexity over a context length l is defined as $P(l) = \exp(\frac{1}{l}\sum_{i=1}^{l} L(i))$. *We warn the readers that the slope of $P(l)$ is less meaningful.* Since $P(l)$ is just the exponential of the cumulative average of $L(i)$, even if $L(i)$ plateaus after some token position k, $P(l)$ may still monotonically decrease after k, giving the wrong impression that the model can make use of the part of the context that is k tokens away.

Results In Figure 2, we show the per-token loss $L(i)$ at different token indices i and perplexity $P(l)$ over different validation context lengths l. As shown in Figure 2, the Forgetting Transformer consistently outperforms the standard Transformer with either architecture, though with the Pro architecture the advantage within the training context length is less significant. Similarly to the standard Transformer, it also maintains a monotonic decreasing per-token loss *within the training context length*, indicating that it utilizes the entire training context for its prediction. In contrast, the per-token loss curves of all recurrent sequence models start flattening at around 5k tokens and completely plateau after 10k tokens. This indicates that these recurrent sequence models struggle to use the full context effectively for their prediction.

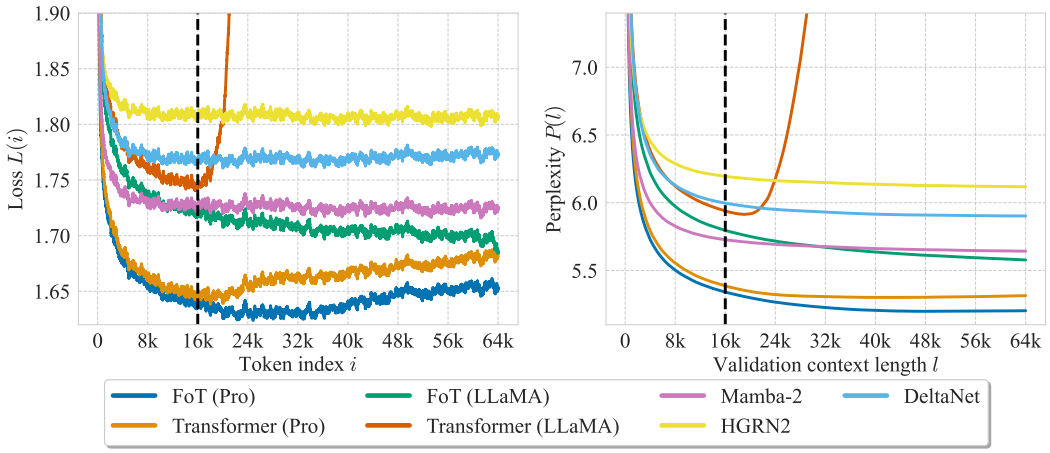

Figure 2: (**left**) Per-token loss $L(i)$ at different token position i. (**right**) Validation perplexity $P(l)$ over different validation context length l. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

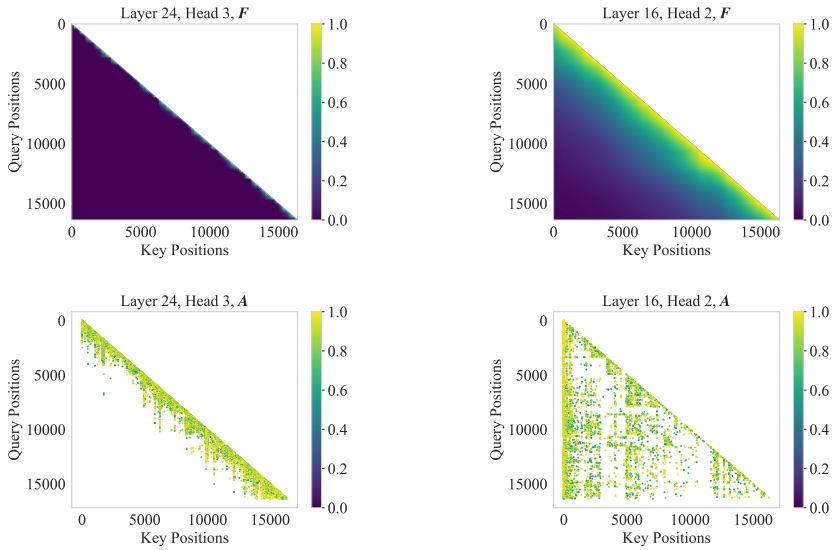

Figure 3: Visualization of the forget gate weight matrix $\boldsymbol{F}$ (top row) and the attention score matrix $\boldsymbol{A}$ (bottom row) from two heads in different layers. Since $\boldsymbol{A}$ is very sparse, we only show entries with scores larger than 0.5. These results use FoT (LLaMA).

The Forgetting Transformer also generalizes beyond the training context length with either architecture. Interestingly, although Transformer (LLaMA) performs poorly beyond the training context length, Transformer (Pro) shows decent length extrapolation. In Appendix E.2 we show that this may be due to the use of QK-norm and output norm. In terms of the *absolute* values of the loss and perplexity, the Forgetting Transformer also clearly outperforms HGRN2 and DeltaNet, and outperforms Mamba-2 when using the Pro architecture.

Visualization of forget gate values and attention map In Figure 3, we visualize the forget gate weight matrix $\boldsymbol{F}$ and the attention scores $\boldsymbol{A} = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top + \log \boldsymbol{F})$ from two heads in different layers. The head on the left-hand side exhibits strong decay, and most entries of $\boldsymbol{F}$ are close to zero; accordingly, the attention focuses on local entries. The head on the right-hand side has much weaker

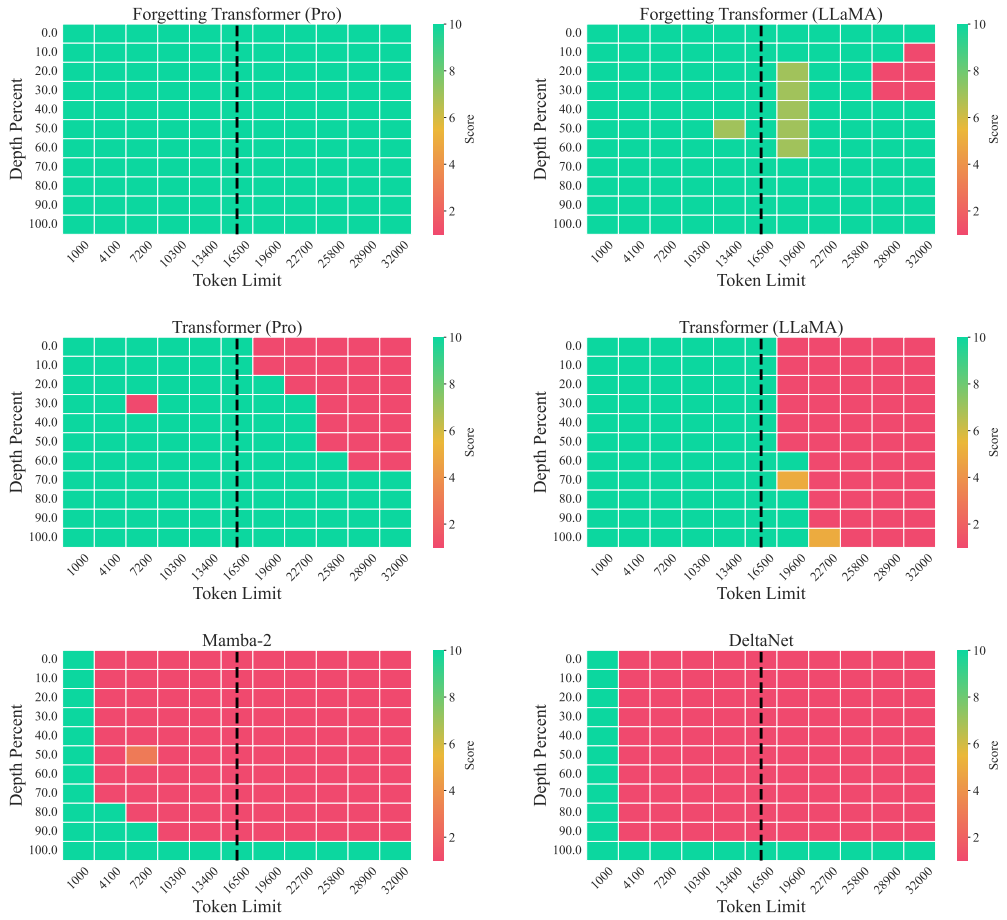

Figure 4: Needle-in-the-haystack analysis for different models. The results are scored on a scale of 1 (red) to 10 (green) by GPT-4o. The vertical dashed line indicates the training context length.

decay, and the attention is distributed across the entire context. This shows that the Forgetting Transformer can learn to retain information across long contexts when necessary.

4.3 Needle in the Haystack

The needle-in-the-haystack analysis (Kamradt, 2023) (referred to as the "needle test" in the following) is a popular test for the long-context retrieval abilities of language models. Following Qin et al. (2024a), we use an "easy mode" of the needle test, where the "needle" placed within the context includes both the question and the answer. This easy mode is particularly suitable for base models that have not been instruction-tuned. Full details, including the prompts used, are in Appendix B.3.

In Figure 4, we show the results of the needle test for Transformer (Pro), FoT (Pro), Transformer (LLaMA), FoT (LLaMA), Mamba2, and DeltaNet. HGRN2 performs even worse than Mamba-2 and we leave it to Appendix E.6. As shown in Figure 4, with either architecture, both FoT and Transformer achieve near-perfect needle retrieval within the training context length, and FoT extrapolates better than the Transformer. In contrast, Mamba-2 and DeltaNet (and also HGRN2 in Appendix E.6) fails even within the training context length, except when the needle is placed right at the end of the text. These results are consistent with the previous analysis of the slope of per-token loss curves in Section 4.2.

Table 1: Evaluation results on LM-eval-harness. All models have roughly 760M non-embedding parameters and are trained on roughly 16B tokens on LongCrawl64. "acc-n" means length-normalized accuracy. Bold and underlined numbers indicate the best and the second best results, respectively.

Model	Wiki. ppl↓	LMB. ppl↓	LMB. acc↑	PIQA acc↑	Hella. acc-n↑	Wino. acc↑	ARC-e acc↑	ARC-c acc-n↑	COPA acc↑	OBQA acc-n↑	SciQA acc↑	BoolQ acc↑	Avg ↑
FoT (Pro)	**27.94**	**21.31**	**39.03**	**61.81**	**33.95**	50.83	**48.78**	**25.68**	62.00	27.00	79.40	**61.13**	**48.96**
Transformer (Pro)	28.54	23.86	37.42	60.83	33.89	50.43	47.85	24.83	65.00	**30.20**	**81.40**	54.16	48.60
FoT (LLaMA)	32.49	28.28	35.55	59.41	32.23	50.51	46.97	23.89	67.00	27.20	77.40	53.76	47.39
Transformer (LLaMA)	32.51	33.94	33.09	60.55	30.89	51.14	44.32	24.23	61.00	27.80	74.90	52.91	46.08
Mamba-2	33.11	42.74	26.80	60.77	32.74	**51.46**	45.71	23.29	**69.00**	28.40	76.30	60.80	47.53
HGRN2	39.27	31.87	33.46	60.12	31.56	49.96	47.60	23.55	63.00	27.20	73.70	42.97	45.31
DeltaNet	35.12	47.49	28.24	60.07	30.83	51.07	46.30	25.26	65.00	28.00	71.40	50.80	45.69

Table 2: Evalution results on LongBench. All models have roughly 760M non-embedding parameters and are trained on roughly 16B tokens on LongCrawl64. Bold and underlined numbers indicate the best and the second best results, respectively.

Model	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Code	
	NarrativeQA	Qasper	MFQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SamSum	LCC	RepoBench-P
FoT (Pro)	10.03	14.44	19.17	**11.1**	16.84	5.44	**22.87**	11.15	**12.87**	56.0	**28.51**	**21.9**	13.11	7.98
Transformer (Pro)	**10.23**	**15.67**	**20.11**	9.81	**21.24**	**5.52**	16.58	10.64	12.78	**63.5**	24.37	19.09	10.29	11.59
FoT (LLaMA)	8.15	13.09	18.51	7.22	16.92	4.3	15.59	12.4	11.51	33.0	14.78	8.28	11.38	13.92
Transformer (LLaMA)	9.32	13.14	16.27	6.96	14.89	5.17	14.86	10.9	12.28	50.0	15.49	10.58	10.33	14.57
Mamba-2	6.63	8.93	16.93	6.39	17.01	3.43	6.89	**13.07**	7.64	11.5	11.64	1.44	15.72	10.38
HGRN2	6.09	7.98	13.26	4.9	12.23	3.06	6.64	9.76	7.54	17.5	12.46	1.06	11.19	16.28
DeltaNet	6.6	7.57	15.25	5.13	12.88	3.21	6.94	10.49	7.9	13.5	13.6	6.04	**17.52**	**18.43**

4.4 Downstream tasks

We evaluate the models on two sets of downstream tasks: a set of short-context tasks from LM-evaluation-harness (Gao et al., 2024) and a set of long-context tasks from LongBench (Bai et al., 2023).

Short-context tasks We use Wikitext (Merity et al., 2016), LAMBADA (Paperno et al., 2016), PiQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Zellers et al., 2019), ARC-easy, ARC-challenge (Clark et al., 2018), Copa (Roemmele et al., 2011), SciQA (Auer et al., 2023), OpenbookQA (Mihaylov et al., 2018), and BoolQA (Clark et al., 2019). Following Yang et al. (2023), we report perplexity for Wikitext and LAMBADA, length-normalized accuracy for HellaSwag, ARC-challenge, and OpenbookQA, and accuracy for all other tasks (we also report accuracy for LAMBADA). All results are zero-shot.

As shown in Table 1, the Forgetting Transformer outperforms the Transformer with either architecture. FoT (Pro) performs the best among all models.

Long-context tasks We use 14 tasks from LongBench: HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), MultiFieldQA-en, NarrativeQA (Kočiskỳ et al., 2018), Qasper (Dasigi et al., 2021), GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), MultiNews (Fabbri et al., 2019), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019), TREC (Li & Roth, 2002), LCC (Guo et al., 2023), and RepoBench-P (Liu et al., 2023). We use the default metrics of LongBench for different tasks, which are either F1, Rough-L, accuracy, or edit similarity.

As shown in Table 2, with either architecture, the Forgetting Transformer performs on par with the Transformer and better than the recurrent sequence models. This provides further evidence that the forget gate does not damage the long-context capabilities of the model.

4.5 Ablation studies

We present two sets of ablation studies. First, we investigate the contribution of each component in FoT and the effect of RoPE. Second, we study the importance of using a forget gate that is data-

Table 3: Ablation experiments for the Forgetting Transformer. All models have roughly 360M non-embedding parameters and are trained on roughly 7.5B tokens on LongCrawl64. The perplexity is measured over a validation context length of 16384 tokens. For the bottom half, all addition (+) or removal (-) of components are relative to FoT (Pro). Per-token loss curves are shown in Appendix E.1.

Model	RoPE	Forget gate	QK-norm	Output gate	Output norm	KV-shift	Perplexity
Transformer (LLaMA) w/o RoPE							14.49
Transformer (LLaMA)	✓						7.52
	✓	✓					7.19
FoT (LLaMA)		✓					7.22
		✓	✓				7.09
		✓	✓	✓			6.89
		✓	✓	✓	✓		6.84
FoT (Pro)		✓	✓	✓	✓	✓	6.65
- QK-norm		✓		✓	✓	✓	6.78
- output gate		✓	✓		✓	✓	6.90
- output norm		✓	✓	✓		✓	6.72
- KV-shift		✓	✓	✓	✓		6.84
+ RoPE	✓	✓	✓	✓	✓	✓	6.65
- forget gate + RoPE (i.e. Transformer (Pro))	✓		✓	✓	✓	✓	6.83
- forget gate			✓	✓	✓	✓	7.37

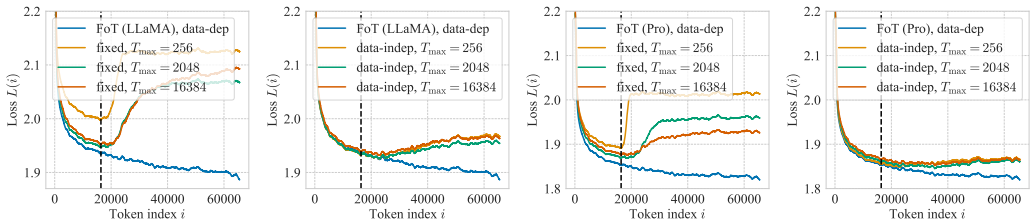

Figure 5: Data-dependent forget gate (data-dep) vs. data-independent (data-indep) and fixed forget gate (i.e., ALiBI). (**left** and **middle-left**) Comparison using the LLaMA architecture. (**middle-right** and **right**) Comparison using the Pro architecture. All per-token loss curves are smoothed by a moving average sliding window of 1001 tokens. The vertical dashed line indicates the training context length.

dependent. For these experiments, we use models with 360M non-embedding parameters trained on roughly 7.5B tokens.

Component analysis We present both (1) an "incremental" style analysis where we incrementally add/remove components from Transformer (LLaMA) to obtain the complete FoT (Pro) model and (2) a "perturbation" style analysis where we add/remove components from FoT (Pro). The results are shown in Table 3. First, as mentioned previously, adding RoPE to FoT (LLaMA) and FoT (Pro) results in minor and no improvement, respectively. Second, both types of analyses show that all components in FoT contribute positively. Also note that with either the LLaMA or the Pro architecture, removing both the forget gate and RoPE results in poor performance (the first and the last row of the table).

Data independent and fixed forget gates To show the importance of using a forget gate that is data-dependent, we test a data-*independent* forget gate $f_t^{(h)} = \sigma(b^{(h)})$, where the superscript (h) means for the h-th head. We also test a forget gate that has fixed values (i.e., $f_t^{(h)} = \sigma(b^{(h)})$, but we do not update $b^{(h)}$ during training). As discussed in Section 3, using a fixed forget gate is equivalent to ALiBi.

For these data-independent forget gate designs, we find it crucial to initialize $b^{(h)}$ properly. To understand the initialization, we first define a function $T(b) = \frac{1}{-\log \sigma(b)}$. This function is defined such that $\sigma(b)^{T(b)} = 1/e$ is always true. We then initialize $b^{(h)} = b_{\text{init}}^{(h)}$ such that $T(b_{(\text{init})}^{(h)}) = \exp(\log T_{\min} + (\log T_{\max} - \log T_{\min})\frac{h-1}{H-1})$, where $T_{\min}$ and $T_{\max}$ are hyperparameters and H

is the number of heads. It can be shown that ALiBi with a maximum slope $\frac{1}{2}$ and a minimum slope $\frac{1}{256}$ (the default values in Press et al. (2021)) is equivalent using a fixed forget gate with $(T_{\min}, T_{\max}) = (2, 256)$. We refer to this initialization as *long-init*.[3] We always set $T_{\min} = 2$.

In Figure 5, we show the per-token loss of different forget gate designs with both the LLaMA and the Pro architecture. As shown in Figure 5, a fixed forget gate (i.e., ALiBi) always underperforms the data-dependent forget gate. With careful initialization, the performance of the data-independent forget gate can approach that of the data-dependent forget gate within the training context length, though it is still worse in terms of length extrapolation.

5 Related Work

Recurrent sequence models While the Transformer has become the de facto standard architecture for sequence modeling, there has been a growing interest in reviving recurrent sequence models (Katharopoulos et al., 2020; Peng et al., 2021; Gu et al., 2021; Orvieto et al., 2023; Yang et al., 2023; Gu & Dao, 2023; Katsch, 2023; De et al., 2024; Sun et al., 2024; Peng et al., 2024; Qin et al., 2024a; Dao & Gu, 2024; Beck et al., 2024; Zhang et al., 2024; Buckman et al., 2024). Many recent recurrent sequence models feature some form of the *forget gate*, which has been shown to be essential in these architectures (Qin et al., 2024b; Gu & Dao, 2023; Yang et al., 2023). Notably, GLA (Yang et al., 2023) and Mamba-2 (Dao & Gu, 2024) show that gated variants of linear attention could be written in a form similar to softmax attention, which directly inspired our work. Several works (Ma et al., 2022; 2024; Ren et al., 2024) combine recurrent layers with quadratic attention. However, unlike our method which embeds the forget gate into the attention mechanism, in these hybrid architectures, the recurrent layers and the quadratic attention layers are independent.

Modifications of softmax attention Several positional embedding methods (Press et al., 2021; Raffel et al., 2020; Chi et al., 2022a;b) add bias to the attention logits depending on the distances between the keys and queries, which can implement data-independent decay. LAS-attention (Zimerman & Wolf) applies multiplicative exponential decay to the attention logits. RoPE (Su et al., 2024) also has a similar decay effect that becomes stronger with increasing relative query/key distances. However, all these methods can only achieve data-*independent* decay based on the relative distances of the queries and keys. CoPE (Olsson et al., 2022) and Selective Attention (Leviathan et al., 2024) modify the attention logit at the current timestep based on cumulative sums of some transformation of attention logits from previous timesteps. Our method is distinct from these in that we calculate forget values separately for the decay instead of reusing the attention logits. Crucially, it is unclear whether CoPE and Selective Attention can be integrated into Flash Attention, which is essential for efficient long-context training.

6 Conclusion

We propose the Forgetting Transformer, a Transformer variant with a forget gate. Our experiments show that the Forgetting Transformer outperforms the standard Transformer and several recurrent sequence models on long-context language modeling and various downstream tasks. We also show that our Transformer Pro block design greatly outperforms the basic LLaMA architecture, with or without a forget gate. We thus recommend future work to adopt FoT (Pro) and Transformer (Pro) as baselines besides the commonly used LLaMA architecture.

We discuss several limitations of our work and potential future work. First, due to our limited computing resources, we can only perform experiments on models up to 760M parameters and a training context length of 16384 tokens. Thus, an important future work is to extend the Forgetting Transformer to larger scales. Second, we only consider causal sequence modeling. It would be interesting to extend the Forgetting Transformer to the non-causal case. Finally, it is possible to perform KV-cache eviction based on the forget gate values, which may greatly reduce inference costs.

[3] We also tested long-init for the data-dependent forget gate but did not find it useful.

REFERENCES

Sören Auer, Dante AC Barone, Cassiano Bartz, Eduardo G Cortes, Mohamad Yaser Jaradeh, Oliver Karras, Manolis Koubarakis, Dmitry Mouromtsev, Dmitrii Pliukhin, Daniil Radyush, et al. The sciqa scientific question answering benchmark for scholarly knowledge. *Scientific Reports*, 13 (1):7240, 2023.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Jacob Buckman. Longcrawl64: A Long-Context Natural-Language Dataset, 2024. URL `https://manifestai.com/articles/longcrawl64`.

Jacob Buckman, Carles Gelada, and Sean Zhang. Symmetric Power Transformers, 2024.

Ta-Chung Chi, Ting-Han Fan, Peter J Ramadge, and Alexander Rudnicky. Kerple: Kernelized relative positional embedding for length extrapolation. *Advances in Neural Information Processing Systems*, 35:8386–8399, 2022a.

Ta-Chung Chi, Ting-Han Fan, Alexander I Rudnicky, and Peter J Ramadge. Dissecting transformer length extrapolation via the lens of receptive field analysis. *arXiv preprint arXiv:2212.10356*, 2022b.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*, 2019.

FlagOpen, 2023. URL https://github.com/FlagOpen/FlagAttention.

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, 07 2024. URL https://zenodo.org/records/12608602.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10): 2222–2232, 2016.

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

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pp. 12098–12107. PMLR, 2023.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*, 2021.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Gregory Kamradt, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack/blob/main/README.md.

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

Tobias Katsch. Gateloop: Fully data-controlled linear recurrence for sequence modeling. *arXiv preprint arXiv:2311.01927*, 2023.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Selective attention improves transformer. *arXiv preprint arXiv:2410.02703*, 2024.

Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

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

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, Lili Yu, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, and Chunting Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. *arXiv preprint arXiv:2404.08801*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

OpenAI, 2021. URL `https://github.com/triton-lang/triton`.

OpenAI, 2022. URL `https://github.com/openai/tiktoken`.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, et al. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 2024.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

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

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024a.

Zhen Qin, Songlin Yang, and Yiran Zhong. Hierarchically gated recurrent neural network for sequence modeling. *Advances in Neural Information Processing Systems*, 36, 2024b.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI spring symposium series*, 2011.

Xuyang Shen, Dong Li, Ruitao Leng, Zhen Qin, Weigao Sun, and Yiran Zhong. Scaling laws for linear complexity language models. *arXiv preprint arXiv:2406.16690*, 2024.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Together Computer. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Jos Van Der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate. *arXiv preprint arXiv:1804.04849*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039*, 2023.

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.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Jiajie Zhang Yuze He Ji Qi Lei Hou Jie Tang Yuxiao Dong Juanzi Li Yushi Bai, Xin Lv. Longalign: A recipe for long context alignment of large language models. *arXiv preprint arXiv:2401.18058*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, et al. Gated slot attention for efficient linear-time sequence modeling. *arXiv preprint arXiv:2409.07146*, 2024.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*, 2021.

Itamar Zimerman and Lior Wolf. Viewing transformers through the lens of long convolutions layers. In *Forty-first International Conference on Machine Learning*.

Appendix

Table of Contents

A Detailed FoT Pro layer computation **17**

B Experimental details **17**
B.1 Model and training hyperparameters 17
B.2 Model parameters, estimated FLOPs, and throughput 17
B.3 Needle in the haystack details 18

C Explanation on the relationship between per-token-loss slope and context utilization **18**

D Hardware-aware implementation of Forgetting Attention **19**

E Additional results **21**
E.1 Per-token loss for the ablation studies 21
E.2 Transformer (Pro) ablation 22
E.3 Short-context training on SlimPajama 22
E.4 Results with 32B training tokens 24
E.5 Comparison with sliding window attention and Samba 25
E.6 Additional needle-in-the-haystack result 26
E.7 Results with 125M-parameter and 360M-parameter models 26
E.8 Effect of QK-norm 26
E.9 Training curves 28
E.10 Stability across random seeds 29
E.11 Impact of parameter initialization 29
E.12 Additional visualization of forget gate matrix 29

A DETAILED FOT PRO LAYER COMPUTATION

We describe the computation of a FoT (Pro) layer (depicted in Figure 1 right) in detail. Each FoT layer takes an input sequence $(\boldsymbol{x}_i)_{i=1}^L$ and produces an output sequence $(\boldsymbol{y}_i)_{i=1}^L$. We first describe the computation for the single head case with head dimension d_{head}. For each time step t, we first compute the key $\boldsymbol{k}_t \in \mathbb{R}^{d_{\text{head}}}$, query $\boldsymbol{q}_t \in \mathbb{R}^{d_{\text{head}}}$, value $\boldsymbol{v}_t \in \mathbb{R}^{d_{\text{head}}}$, forget gate $f_t \in \mathbb{R}$, and output gate $\boldsymbol{g}_t \in \mathbb{R}^{d_{\text{head}}}$ as follows:

$$\boldsymbol{q}_t = \mathrm{RMSNorm}(\boldsymbol{W}_q \boldsymbol{x}_t) \tag{16}$$

$$\tilde{\boldsymbol{k}}_t = \boldsymbol{W}_k \boldsymbol{x}_t, \quad \alpha_t^{\text{key}} = \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_t) \in \mathbb{R} \tag{17}$$

$$\boldsymbol{k}_t = \mathrm{RMSNorm}(\alpha_t^{\text{key}} \tilde{\boldsymbol{k}}_{t-1} + (1 - \alpha_t^{\text{key}}) \tilde{\boldsymbol{k}}_t) \tag{18}$$

$$\tilde{\boldsymbol{v}}_t = \boldsymbol{W}_k \boldsymbol{x}_t, \quad \alpha_t^{\text{value}} = \sigma(\boldsymbol{w}_v^\top \boldsymbol{x}_t) \in \mathbb{R} \tag{19}$$

$$\boldsymbol{v}_t = \alpha_t^{\text{value}} \tilde{\boldsymbol{v}}_{t-1} + (1 - \alpha_t^{\text{value}}) \tilde{\boldsymbol{v}}_t \tag{20}$$

$$f_t = \sigma(\boldsymbol{w}_f^\top \boldsymbol{x}_t + b_f) \in \mathbb{R} \tag{21}$$

$$\boldsymbol{g}_t = \boldsymbol{W}_g \boldsymbol{x}_t. \tag{22}$$

This is followed by the computation of the attention output:

$$\boldsymbol{o}_i = \frac{\sum_{j=1}^i \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j + d_{ij}) \boldsymbol{v}_j}{\sum_{j=1}^i \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j + d_{ij})} \tag{23}$$

where $d_{ij} = \sum_{l=j+1}^i \log f_l$. We then apply the output normalization, output gate, and the final output projection:

$$\boldsymbol{y}_i = \boldsymbol{W}_o(\mathrm{RMSNorm}(\boldsymbol{o}_i) \odot \boldsymbol{g}_i). \tag{24}$$

For the multi-head case, each head h maintains an independent copy of the parameters and computes its output sequence $(\boldsymbol{y}_i^{(h)})_{i=1}^L$ independently. $\boldsymbol{y}_i^{(h)}$'s from different heads are then summed together to obtain the final output $\boldsymbol{y}_i$. Note that similar to the standard Transformer, even though the computation and parameters of different heads are conceptually independent, the computation can be implemented equivalently by properly splitting/concatenating the intermediate vectors/weight matrices of different heads.

B EXPERIMENTAL DETAILS

B.1 MODEL AND TRAINING HYPERPARAMETERS

All models in the main experiment have roughly 760M non-embedding parameters. We do not share the embedding parameters with the last linear layer. All models have a hidden dimension of 1536, a head dimension of 128, and 24 blocks of multi-head attention + MLP layers. As mentioned in the main text, we use $\theta = 500000$ for RoPE. For other hyperparameters including initialization methods, we use the default settings in Flash Linear Attention (Yang & Zhang, 2024).

For the ablation experiments in Section 4.5, all models have roughly 360M non-embedding parameters. The hidden dimension is 1024. For other model hyperparameters, we use the default values in Flash Linear Attention (Yang & Zhang, 2024). We use a linear learning rate warmup from 0 to 1.5×10^{-3} for the first 256×2^{20} tokens and then a cosine decay schedule to 1.5×10^{-4}. Other training-related hyperparameters are the same as the 760M-parameter setting except for the Transformer we use a head dimension of 64.

B.2 MODEL PARAMETERS, ESTIMATED FLOPS, AND THROUGHPUT

We report the exact number of (non-embedding) parameters, estimated FLOPs and throughput in Table 4. Note that the attention kernels in FoT (Pro), Transformer (Pro) and FoT (LLaMA) are implemented in Triton by us while Transformer (LLaMA) uses the official Flash Attention implementation (Dao, 2023)) implemented in CUDA. We expect these four models to have similar throughput if their kernels are all implemented in CUDA.

Model	Parameters	estimated FLOPs/token	Throughput (token/sec)
FoT (Pro)	758M	8.18×10^9	38k
Transformer (Pro)	757M	8.18×10^9	38k
FoT (LLaMA)	757M	8.18×10^9	47k
Transformer (LLaMA)	756M	8.18×10^9	60k
Mamba-2	780M	4.96×10^9	72k
HGRN2	756M	4.63×10^9	70k
DeltaNet	757M	4.64×10^9	70k

Table 4: Comparison of different models in terms of number of parameters, estimated FLOPs per token, and model throughput. We count non-embedding parameters as the embedding parameters' contribution to computational costs are negligible. For the estimated FLOPs, we omit sub-leading terms such as RMSNorm and residual connections. For the FLOPs of recurrent models, we assume they use the most FLOP-efficient sequential recurrent form of computation and thus the estimation is a strict lower bound of the FLOPs in practice. Throughputs are measured on 4 A100-80GB GPUs.

Since we use a very long training context length, recurrent models have a huge advantage in theoretical FLOPs due to their linear complexity. Though an exact FLOPs-matched comparison would be interesting, it will require recalibrating the scaling law for the long-context setting and is beyond the scope of this work. Nevertheless, empirical evidence in the literature suggests it is unlikely that using larger models will qualitatively change the poor long-context capabilities of recurrent models. For example, Qin et al. (2024a) trains 3B-parameter HGRN2 and Mamba and both still perform poorly in the needle-in-the-haystack task.

B.3 Needle in the haystack details

We use the needle test in the LongAlign (Yushi Bai, 2024), which is adapted from the original needle test reposiroty (Kamradt, 2023) for HuggingFace[4] models. The prompt has the following structure:

```
[irrelevant context...]
What is the best thing to do in San Francisco? Answer: The best thing to
do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny
day.
[irrelevant context...]

There is an important piece of information hidden inside the above
document. Now that you've read the document, I will quiz you about it.
Answer the following question: What is the best thing to do in San
Francisco? Answer:
```

The results are scored by GPT-4o on a scale from 1 to 10.

C Explanation on the relationship between per-token-loss slope and context utilization

To understand the relationship between the slope of the per-token loss and context utilization of the model, we first point out that LongCrawl64 applies the preprocessing of randomly "rolling" the sequences[5] to remove any position bias. This means that *when given contexts of the same length*, the difficulty of predicting tokens at different positions is roughly the same on average. For example, predicting the 100-th tokens in the sequences *only given the previous* 90 *tokens* is roughly as difficult as predicting the 90-th tokens when given the full previous 90-token context. Therefore, if $L(100) < L(90)$, it indicates that the first 10 tokens in the context contribute to the model's predictions for the 100-th token; and larger the difference $L(90) - L(100)$ is, the more these distant tokens contribute.

[4] `https://huggingface.co/`

[5] Concretely, this can be implemented with `np.roll` with random shift value

On the other hand, if $L(100)$ is roughly the same $L(90)$ (i.e., the graph of $L(i)$ plateaus after $i = 100$), it means the first 10 tokens do not contribute to the model's prediction for the 100-th token, either because they are inherently not useful for this prediction or the model are unable to utilize them.

In summary, the slope of $L(i)$ at token position i reflects how much tokens from roughly i steps earlier contribute to the model's prediction at the current token position.

D Hardware-aware implementation of Forgetting Attention

Algorithm 1 Forgetting Attention Forward Pass

Require: Matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times d}$, vector $\boldsymbol{c} \in \mathbb{R}^N$ in HBM, block sizes B_c, B_r.

Divide $\boldsymbol{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\boldsymbol{K}, \boldsymbol{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\boldsymbol{K}_1, \dots, \boldsymbol{K}_{T_c}$ and $\boldsymbol{V}_1, \dots, \boldsymbol{V}_{T_c}$, of size $B_c \times d$ each.

Divide the output $\boldsymbol{O} \in \mathbb{R}^{N \times d}$ into T_r blocks $\boldsymbol{O}_1, \dots, \boldsymbol{O}_{T_r}$ of size $B_r \times d$ each, and divide the logsumexp L into T_r blocks $L_1, \dots, L_{T_r}$ of size B_r each.

Let $\boldsymbol{c}^q = \boldsymbol{c}$. Devide $\boldsymbol{c}^q$ into T_r blocks $\boldsymbol{c}^q_1, \dots, \boldsymbol{c}^q_{T_r}$

Let $\boldsymbol{c}^k = \boldsymbol{c}$. Devide $\boldsymbol{c}^k$ into T_c blocks $\boldsymbol{c}^k_1, \dots, \boldsymbol{c}^k_{T_c}$

for $1 \le i \le T_r$ **do**

  Load $\boldsymbol{Q}_i$, $\boldsymbol{c}^q_i$ from HBM to on-chip SRAM.

  On chip, initialize $\boldsymbol{O}_i^{(0)} = (0)_{B_r \times d} \in \mathbb{R}^{B_r \times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.

  for $1 \le j \le T_c$ **do**

    Load $\boldsymbol{K}_j, \boldsymbol{V}_j$, $\boldsymbol{c}^k_j$ from HBM to on-chip SRAM.

    On chip, compute $\boldsymbol{S}_i^{(j)} = \boldsymbol{Q}_i \boldsymbol{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{D}_i^{(j)} = \boldsymbol{c}^q_i \mathbf{1}^\top - \mathbf{1}(\boldsymbol{c}^k_j)^\top \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{S}_i^{(j)} = \boldsymbol{S}_i^{(j)} + \boldsymbol{D}_i^{(j)} \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{S}_i^{(j)} = \mathrm{mask}(\boldsymbol{S}_i^{(j)}, i, j) \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\boldsymbol{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\boldsymbol{P}}_i^{(j)} = \exp(\boldsymbol{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} \ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\boldsymbol{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.

    On chip, compute $\boldsymbol{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} \boldsymbol{O}_i^{(j-1)} + \tilde{\boldsymbol{P}}_i^{(j)} \boldsymbol{V}_j$.

  end for

  On chip, compute $\boldsymbol{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1} \boldsymbol{O}_i^{(T_c)}$.

  On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.

  Write $\boldsymbol{O}_i$ to HBM as the i-th block of $\boldsymbol{O}$.

  Write L_i to HBM as the i-th block of L.

end for

Return the output $\boldsymbol{O}$ and the logsumexp L.

In Algorithm 1 and 2, we provide the algorithm for computing the forward pass and backward pass of Forgetting Attention in a hardware-aware way. The algorithm is reproduced from Flash Attention 2 (Dao, 2023), with the changes needed to implement Forgetting Attention added and highlighted. In this algorithm, we assume that we pre-computed the cumulative sum $\boldsymbol{c} = \mathrm{cumsum}(\log \boldsymbol{f})$. The mask operation properly sets some entries of its first operand to $-\infty$ to satisfy the causality requirement. Note for the backward pass for ease of presentation we combine the computation of $\mathbf{d}\boldsymbol{K}, \mathbf{d}\boldsymbol{V}, \mathbf{d}\boldsymbol{c}^k$ and the computation of $\mathbf{d}\boldsymbol{Q}, \mathbf{d}\boldsymbol{c}^q$ in a single algorithm, but in practice these are computed in two different kernels for implementation simplicity.

Algorithm 2 Forgetting Attention Backward Pass

Require: Matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{O}, \mathbf{d}\boldsymbol{O} \in \mathbb{R}^{N \times d}$ in HBM, vector $\boldsymbol{c}, \mathbf{d}\boldsymbol{c} \in \mathbb{R}^{N}$, vector $L \in \mathbb{R}^{N}$ in HBM, block sizes B_c, B_r.

Divide $\boldsymbol{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\boldsymbol{K}, \boldsymbol{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\boldsymbol{K}_1, \dots, \boldsymbol{K}_{T_c}$ and $\boldsymbol{V}_1, \dots, \boldsymbol{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\boldsymbol{O}$ into T_r blocks $\boldsymbol{O}_1, \dots, \boldsymbol{O}_{T_r}$ of size $B_r \times d$ each, divide $\mathbf{d}\boldsymbol{O}$ into T_r blocks $\mathbf{d}\boldsymbol{O}_1, \dots, \mathbf{d}\boldsymbol{O}_{T_r}$ of size $B_r \times d$ each, and divide L into T_r blocks $L_i, \dots, L_{T_r}$ of size B_r each.

Initialize $\mathbf{d}\boldsymbol{Q} = (0)_{N \times d}$ in HBM and divide it into T_r blocks $\mathbf{d}\boldsymbol{Q}_1, \dots, \mathbf{d}\boldsymbol{Q}_{T_r}$ of size $B_r \times d$ each. Divide $\mathbf{d}\boldsymbol{K}, \mathbf{d}\boldsymbol{V} \in \mathbb{R}^{N \times d}$ in to T_c blocks $\mathbf{d}\boldsymbol{K}_1, \dots, \mathbf{d}\boldsymbol{K}_{T_c}$ and $\mathbf{d}\boldsymbol{V}_1, \dots, \mathbf{d}\boldsymbol{V}_{T_c}$, of size $B_c \times d$ each.

Let $\boldsymbol{c}^q = \boldsymbol{c}^k = \boldsymbol{c}$. Devide $\boldsymbol{c}^q$ into T_r blocks $\boldsymbol{c}^q_1, \dots, \boldsymbol{c}^q_{T_r}$. Devide $\boldsymbol{c}^k$ into T_c blocks $\boldsymbol{c}^k_1, \dots, \boldsymbol{c}^k_{T_c}$.

Let $\mathbf{d}\boldsymbol{c}^q = \mathbf{d}\boldsymbol{c}^k = (0)_N$. Devide $\mathbf{d}\boldsymbol{c}^q$ into T_r blocks $\mathbf{d}\boldsymbol{c}^q_1, \dots, \mathbf{d}\boldsymbol{c}^q_{T_r}$. Devide $\mathbf{d}\boldsymbol{c}^k$ into T_c blocks $\mathbf{d}\boldsymbol{c}^k_1, \dots, \mathbf{d}\boldsymbol{c}^k_{T_c}$.

Compute $D = \mathrm{rowsum}(\mathbf{d}\boldsymbol{O} \circ \boldsymbol{O}) \in \mathbb{R}^d$ (pointwise multiply), write D to HBM and divide it into T_r blocks $D_1, \dots, D_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\boldsymbol{K}_j, \boldsymbol{V}_j, \boldsymbol{c}^k_j$ from HBM to on-chip SRAM.

  Initialize $\mathbf{d}\boldsymbol{K}_j = (0)_{B_c \times d}, \mathbf{d}\boldsymbol{V}_j = (0)_{B_c \times d}$ on SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\boldsymbol{Q}_i, \boldsymbol{O}_i, \mathbf{d}\boldsymbol{O}_i, \mathbf{d}\boldsymbol{Q}_i, L_i, D_i, \boldsymbol{c}^q_i$ from HBM to on-chip SRAM.

    On chip, compute $\boldsymbol{S}^{(j)}_i = \boldsymbol{Q}_i \boldsymbol{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{D}^{(j)}_i = \boldsymbol{c}^q_i \mathbf{1}^\top - \mathbf{1}(\boldsymbol{c}^k_j)^\top \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{S}^{(j)}_i = \boldsymbol{S}^{(j)}_i + \boldsymbol{D}^{(j)}_i \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{S}^{(j)}_i = \mathrm{mask}(\boldsymbol{S}^{(j)}_i, i, j) \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\boldsymbol{P}^{(j)}_i = \exp(\boldsymbol{S}_{ij} - L_i) \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\mathbf{d}\boldsymbol{V}_j \leftarrow \mathbf{d}\boldsymbol{V}_j + (\boldsymbol{P}^{(j)}_i)^\top \mathbf{d}\boldsymbol{O}_i \in \mathbb{R}^{B_c \times d}$.

    On chip, compute $\mathbf{d}\boldsymbol{P}^{(j)}_i = \mathbf{d}\boldsymbol{O}_i \boldsymbol{V}_j^\top \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\mathbf{d}\boldsymbol{S}^{(j)}_i = \boldsymbol{P}^{(j)}_i \circ (\mathbf{d}\boldsymbol{P}^{(j)}_i - D_i) \in \mathbb{R}^{B_r \times B_c}$.

    Load $\mathbf{d}\boldsymbol{Q}_i$ from HBM to SRAM, then on chip, update $\mathbf{d}\boldsymbol{Q}_i \leftarrow \mathbf{d}\boldsymbol{Q}_i + \mathbf{d}\boldsymbol{S}^{(j)}_i \boldsymbol{K}_j \in \mathbb{R}^{B_r \times d}$, and write back to HBM.

    Load $\mathbf{d}\boldsymbol{c}^q_i$ from HBM to SRAM, then on chip, update $\mathbf{d}\boldsymbol{c}^q_i \leftarrow \mathbf{d}\boldsymbol{c}^q_i + \mathbf{d}\boldsymbol{S}^{(j)}_i \mathbf{1} \in \mathbb{R}^{B_r}$, and write back to HBM.

    On chip, compute $\mathbf{d}\boldsymbol{K}_j \leftarrow \mathbf{d}\boldsymbol{K}_j + \mathbf{d}\boldsymbol{S}^{(j)\top}_i \boldsymbol{Q}_i \in \mathbb{R}^{B_c \times d}$.

    On chip, compute $\mathbf{d}\boldsymbol{c}^k_j \leftarrow \mathbf{d}\boldsymbol{c}^k_j - \mathbf{d}\boldsymbol{S}^{(j)\top}_i \mathbf{1} \in \mathbb{R}^{B_c}$.

  end for

  Write $\mathbf{d}\boldsymbol{K}_j, \mathbf{d}\boldsymbol{V}_j, \mathbf{d}\boldsymbol{c}^k_j$ to HBM.

end for

Compute $\mathbf{d}\boldsymbol{c} = \mathbf{d}\boldsymbol{c}^q + \mathbf{d}\boldsymbol{c}^k$.

Return $\mathbf{d}\boldsymbol{Q}, \mathbf{d}\boldsymbol{K}, \mathbf{d}\boldsymbol{V}, \mathbf{d}\boldsymbol{c}$.

In practice, we implement Forgetting Attention based on the Triton (OpenAI, 2021) Flash Attention implementation in Flag Attention (FlagOpen, 2023).

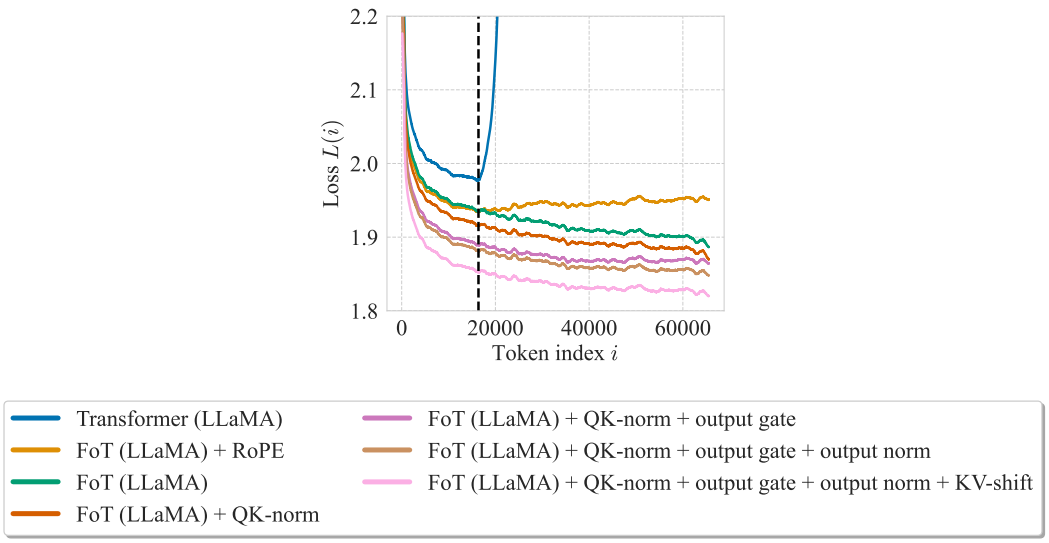

Figure 6: Per-token loss for the incremental-style ablation studies presented in Section 4.5. All models are roughly 360M parameters and are trained on roughly 7.5B tokens on LongCrawl64. The vertical line indicates the training context length.

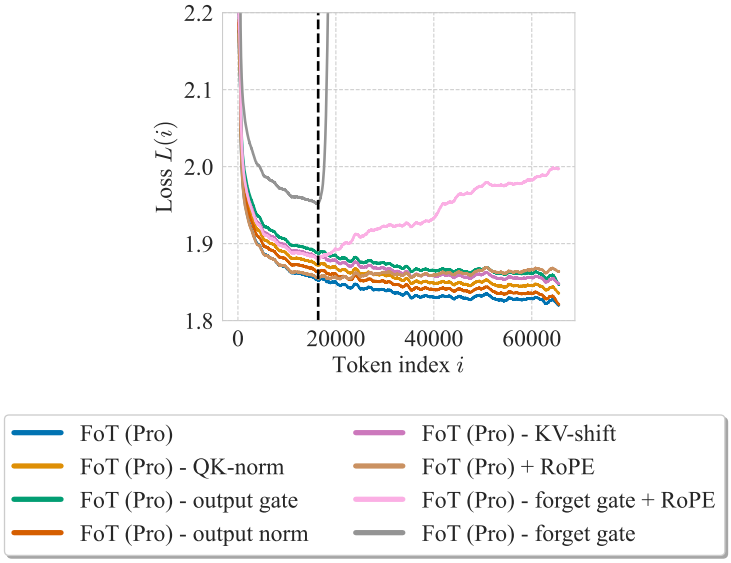

Figure 7: Per-token loss for the perturbation-style ablation studies presented in Section 4.5. All models are roughly 360M parameters and are trained on roughly 7.5B tokens on LongCrawl64. The vertical line indicates the training context length.

E Additional results

E.1 Per-token loss for the ablation studies

In Figure 6 and Figure 7 we show the per-token loss for the ablation studies presented in Table 3 in Section 4.5. Transformer (LLaMA) without RoPE performs extremely poorly and we show it separately in Figure 8.

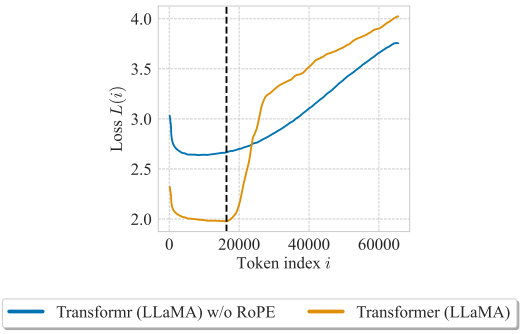

Figure 8: Removing RoPE from Transformer (LLaMA) results in poor performance. All models are roughly 360M parameters and are trained on roughly 7.5B tokens on LongCrawl64. The vertical line indicates the training context length.

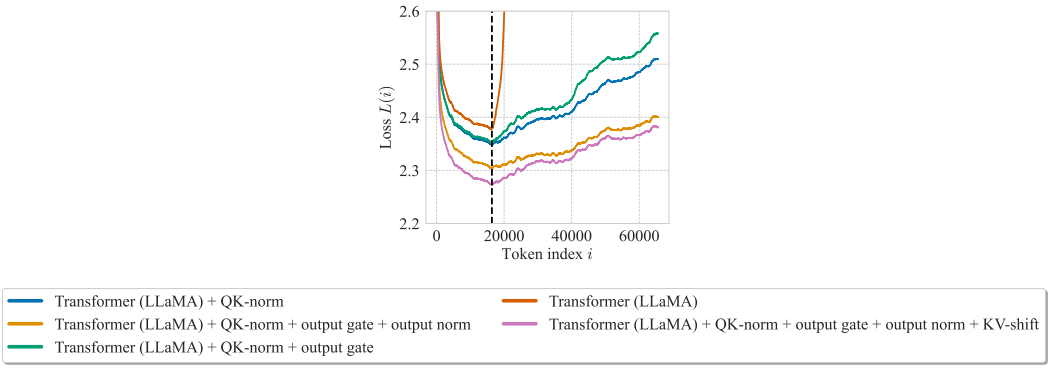

Figure 9: Incremental style ablation study for Transformer (Pro). All models are roughly 125M parameters and are trained on roughly 2.5B tokens on LongCrawl64. The vertical line indicates the training context length.

E.2 Transformer (Pro) ablation

In Figure 9 we present a small-scale ablation study using Transformer (Pro) in the 125M-parameter/2.5B-token setting. We start from Transformer (LLaMA) and add one component at a time. Notably, we find that QK-norm and output norm are helpful for length extrapolation.

E.3 Short-context training on SlimPajama

To complement our main results in which we perform long-context training on LongCrawl64, we have also run short-context training on the more commonly used SlimPajama dataset (Soboleva et al., 2023). We follow the 340M-parameter/15B-token/2k-context-length setting in Yang et al. (2024). We also use the same hyperparameters and tokenizer as Yang et al. (2024). We train FoT and Transformer with both the LLaMA and the Pro architecture. We also test Mamba-2, the strongest recurrent sequence model in our main results.

We show the per-token loss of tested models in Figure 10 and downstream task evaluation results in Table 5. We use the same set of tasks as Yang et al. (2024) so our results can be directly compared to those of Yang et al. (2024). As shown in the results, in this short-context training setting FoT does not have an advantage over the Transformer in terms of language modeling loss within the training context length. However, it still shows better length extrapolation and superior downstream task performance.

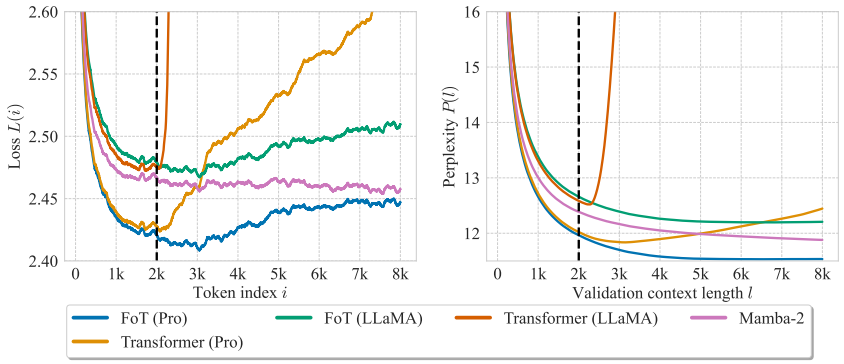

Figure 10: Results on SlimPajama with a training context length of 2048 tokens. All models have roughly 340M non-embedding parameters and are trained on roughly 15B tokens on SlimPajama. The vertical line indicates the training context length.

Table 5: Evaluation results on LM-eval-harness for models trained on SlimPajama with a training context length of 2048 tokens. All models have roughly 340M non-embedding parameters and are trained on roughly 15B tokens on SlimPajama. "acc-n" means length-normalized accuracy. Bold and underlined numbers indicate the best and the second best results, respectively. The results for Transformers++ and DeltaNet are from Yang et al. (2024). Note that Transformer++ from Yang et al. (2024) and Transformer (LLaMA) in our work have exactly the same architecture

Model	**Wiki.** ppl↓	**LMB.** ppl↓	**LMB.** acc↑	**PIQA** acc↑	**Hella.** acc-n↑	**Wino.** acc↑	**ARC-e** acc↑	**ARC-c** acc-n↑	**Avg** ↑
Transformer++ (Yang et al., 2024)	28.39	42.69	31.00	63.30	34.00	50.40	44.50	<u>24.20</u>	41.23
DeltaNet (Yang et al., 2024)	28.24	37.37	32.10	64.80	34.30	**52.20**	<u>45.80</u>	23.20	42.07
FoT (Pro)	**25.69**	<u>31.98</u>	**35.82**	<u>65.61</u>	**36.39**	<u>51.07</u>	45.79	**25.09**	**43.29**
Transformer (Pro)	<u>25.92</u>	**31.93**	<u>35.01</u>	65.02	<u>36.09</u>	50.51	**46.42**	23.38	<u>42.74</u>
FoT (LLaMA)	27.86	43.26	32.56	64.80	34.59	50.12	45.12	23.38	41.76
Transformer (LLaMA)	27.98	35.25	32.31	63.71	34.89	48.07	45.33	23.72	41.34
Mamba-2	27.51	41.32	29.83	**65.94**	35.95	50.20	45.45	23.72	41.85

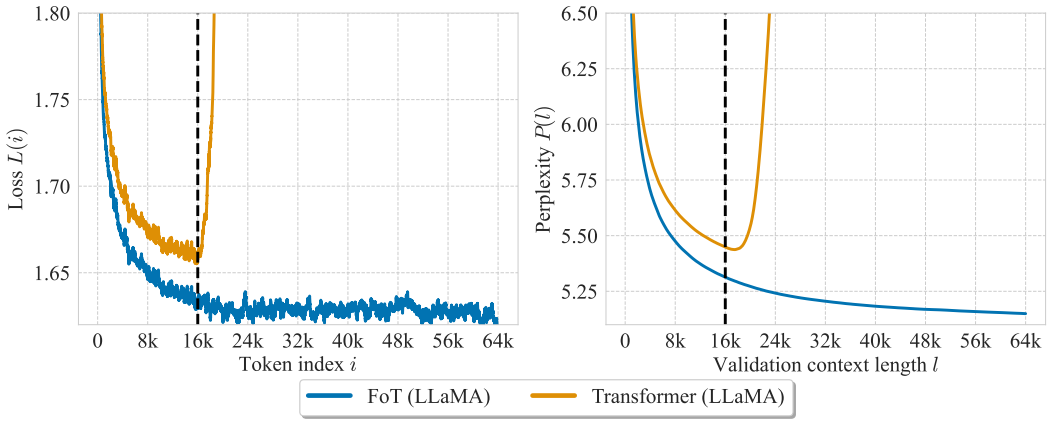

Figure 11: Comparison with 32B training tokens. (**left**) Per-token loss $L(i)$ at different token position i. (**right**) Validation perplexity $P(l)$ over different validation context length l. All models have 760M parameters and are trained on roughly 32B tokens. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

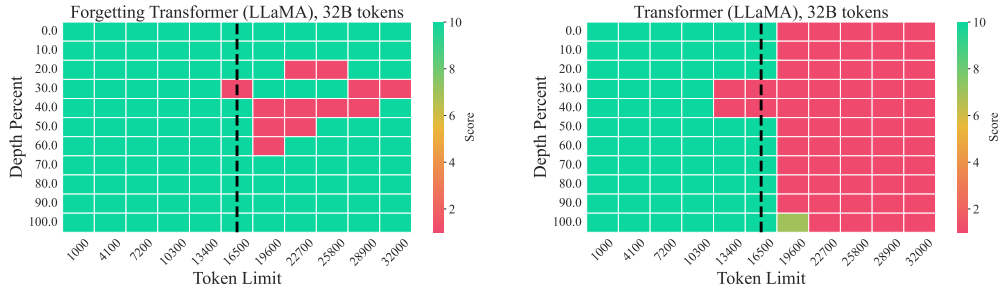

Figure 12: Needle-in-the-haystack analysis for different models trained with 32B tokens. The results are scored on a scale of 1 (red) to 10 (green) by GPT-4o. The vertical dashed line indicates the training context length.

E.4 Results with 32B training tokens

To verify that our results remain consistent with longer training runs, we have trained FoT (LLamA) and Transformer (LLaMA) with 32B tokens. We show the per-token loss, needle-retrieval experiment, short-context downstream task results, and long-context task results in Figure 11, Figure 12, Table 6, and Table 7 respectively. These results are consistent with those obtained with the 16B-token training runs in the main text, though longer training seems to damage needle retrieval performance beyond the training context length to some extent.

Table 6: Evaluation results on LM-eval-harness with 32B training tokens and 760M non-embedding-parameter models. "acc-n" means length-normalized accuracy. Bold and underlined numbers indicate the best and the second best results, respectively.

Model	Wiki. ppl↓	LMB. ppl↓	LMB. acc↑	PIQA acc↑	Hella. acc-n↑	Wino. acc↑	ARC-e acc↑	ARC-c acc-n↑	COPA acc↑	OBQA acc-n↑	SciQA acc↑	BoolQ acc↑	Avg ↑
FoT (LLaMA)	**29.46**	**22.40**	**37.69**	**61.92**	**33.91**	**51.22**	**47.77**	23.72	**65.00**	**28.20**	**81.10**	49.88	**48.04**
Transformer (LLaMA)	29.57	24.21	36.52	61.32	32.84	48.38	46.21	**24.74**	60.00	27.60	79.10	**61.56**	47.83

Table 7: Evaluation results on LongBench with 32B training tokens and 760M non-embedding-parameter models. Bold and underlined numbers indicate the best and the second best results, respectively.

Model	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Code	
	NarrativeQA	Qasper	MFQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SamSum	LCC	RepoBench-P
FoT (LLaMA)	**10.62**	12.17	**20.96**	6.53	17.82	**5.86**	**19.4**	**11.63**	8.74	**48.0**	**25.7**	**21.58**	5.9	10.55
Transformer (LLaMA)	9.62	**12.52**	19.65	**9.63**	**18.53**	4.31	15.45	10.85	**9.15**	43.5	19.9	20.66	**11.98**	**17.43**

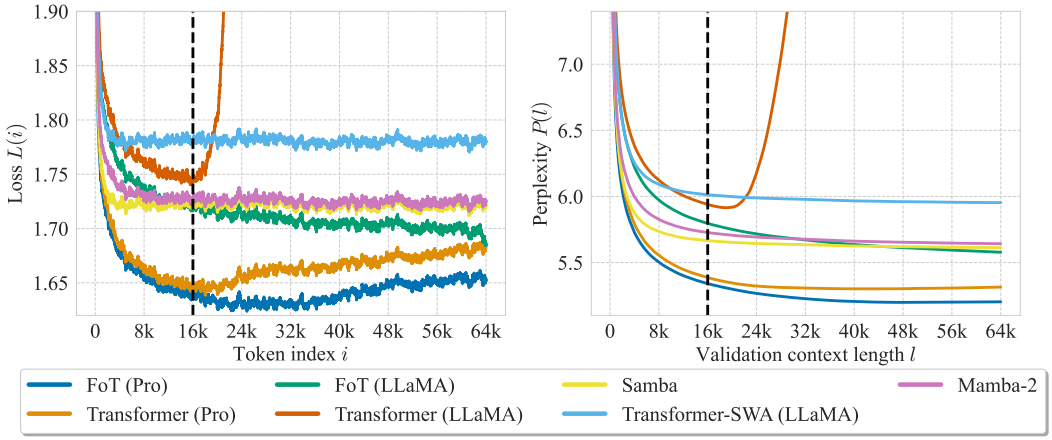

Figure 13: Additional comparison with Samba and Transformer-SWA. (**left**) Per-token loss $L(i)$ at different token position i. (**right**) Validation perplexity $P(l)$ over different validation context length l. All models have 760M parameters and are trained on roughly 16B tokens. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

E.5 Comparison with sliding window attention and Samba

In this Section, we compare the standard Transformer and FoT with a sliding-window-attention-based Transformer (Transformer-SWA). We also compare with Samba (Ren et al., 2024), a hybrid architecture combining sliding window attention and Mamba. Both Transformer-SWA and Samba use a window size of 2048. We show the per-token loss, needle-retrieval experiment, short-context downstream task results, and long-context task results in Figure 13, Figure 14, Table 8, and Table 9 respectively. Though both Transformer-SWA and Samba perform well on short-context tasks, they show early plateau in the per-token loss, which indicates that they struggle to utilize the long context. Accordingly, they perform poorly in the needle-retrieval task.

Table 8: Evaluation results on LM-eval-harness including Samba and Transformer-SWA. All models have roughly 760M non-embedding parameters and are trained on roughly 16B tokens on LongCrawl64. "acc-n" means length-normalized accuracy. Bold and underlined numbers indicate the best and the second best results, respectively.

Model	Wiki. ppl↓	LMB. ppl↓	LMB. acc↑	PIQA acc↑	Hella. acc-n↑	Wino. acc↑	ARC-e acc↑	ARC-c acc-n↑	COPA acc↑	OBQA acc-n↑	SciQA acc↑	BoolQ acc↑	Avg ↑
FoT (Pro)	**27.94**	**21.31**	**39.03**	61.81	**33.95**	50.83	**48.78**	**25.68**	62.00	27.00	79.40	**61.13**	48.96
Transformer (Pro)	28.54	23.86	37.42	60.83	33.89	50.43	47.85	24.83	65.00	**30.20**	**81.40**	54.16	48.60
FoT (LLaMA)	32.49	28.28	35.55	59.41	32.23	50.51	46.97	23.89	**67.00**	27.20	77.40	53.76	47.39
Transformer (LLaMA)	32.51	33.94	33.09	60.55	30.89	**51.14**	44.32	24.23	61.00	27.80	74.90	52.91	46.08
Samba	32.22	26.28	35.36	**62.02**	32.89	51.07	48.70	25.09	66.00	28.80	79.10	**61.13**	**49.02**
Transformer-SWA (LLaMA)	34.13	33.61	33.15	59.79	32.03	48.22	46.59	25.34	66.00	26.80	77.10	60.12	47.51

Table 9: Evalution results on LongBench including Samba and Transformer-SWA. All models have roughly 760M non-embedding parameters and are trained on roughly 16B tokens on LongCrawl64. Bold and underlined numbers indicate the best and the second best results, respectively.

Model	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Code	
	NarrativeQA	Qasper	MFQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SamSum	LCC	RepoBench-P
FoT (Pro)	10.03	14.44	19.17	**11.1**	16.84	5.44	**22.87**	11.15	**12.87**	56.0	**28.51**	**21.9**	**13.11**	7.98
Transformer (Pro)	**10.23**	**15.67**	**20.11**	9.81	**21.24**	**5.52**	16.58	10.64	12.78	**63.5**	24.37	19.09	10.29	11.59
FoT (LLaMA)	8.15	13.09	18.51	7.22	16.92	4.3	15.59	**12.4**	11.51	33.0	14.78	8.28	11.38	13.92
Transformer (LLaMA)	9.32	13.14	16.27	6.96	14.89	5.17	14.86	10.9	12.28	50.0	15.49	10.58	10.33	**14.57**
Samba	8.85	7.2	15.77	8.94	16.78	3.11	8.35	10.23	11.84	30.0	17.62	4.4	11.8	11.79
Transformer-SWA (LLaMA)	7.89	11.07	15.14	7.17	13.27	2.81	9.97	8.97	11.18	39.0	18.28	4.97	9.73	11.0

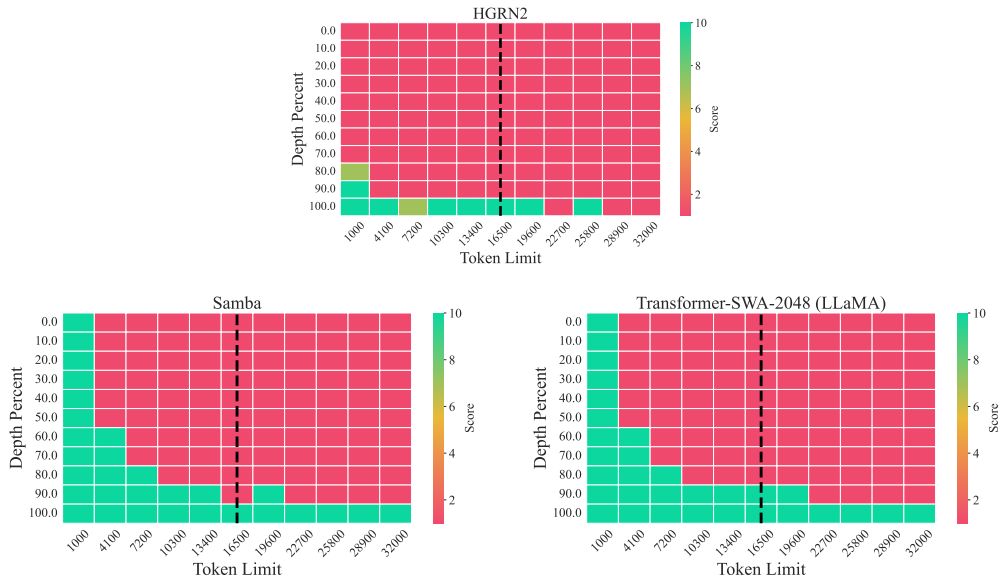

Figure 14: Needle-in-the-haystack analysis for HGRN2, Samba, and Transformer with sliding window attention (SWA). The results are scored on a scale of 1 (red) to 10 (green) by GPT-4o. The vertical dashed line indicates the training context length.

E.6 Additional needle-in-the-haystack result

In Figure 14, we show the results of the needle test for HGRN2, Samba, and Transformer-SWA (LLaMA). Samba and Transformer-SWA use a sliding window of 2048 tokens.

E.7 Results with 125M-parameter and 360M-parameter models

In Figure 15 and Figure 16 we show the per-token loss for 125M-parameter models trained on roughly 2.5B tokens and 360M-parameter models trained on roughly 7.5B tokens, respectively. These results are consistent with the 760M-parameter/16B-token results in Figure 2.

E.8 Effect of QK-norm

In Figure 17 we show the results for FoT (LLaMA) + QK-norm and Transformer (LLaMA) + QK-norm. We see that QK-norm is equally useful for both models. Interestingly, it improves the length extrapolation of Transformer (LLaMA). We conjecture that this is because QK-norm bounds the L2-norm of the queries and keys and thus the model will be more robust to unusual activation values due to extrapolation to unseen context lengths.

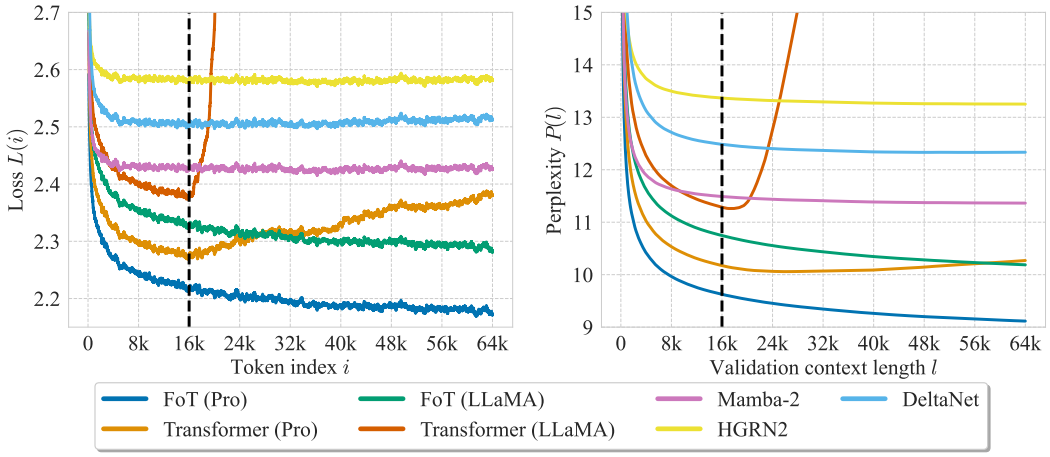

Figure 15: Results with 125M-parameter models trained on roughly 2.5B tokens. (**left**) Per-token loss $L(i)$ at different token position i. (**right**) Validation perplexity $P(l)$ over different validation context length l. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

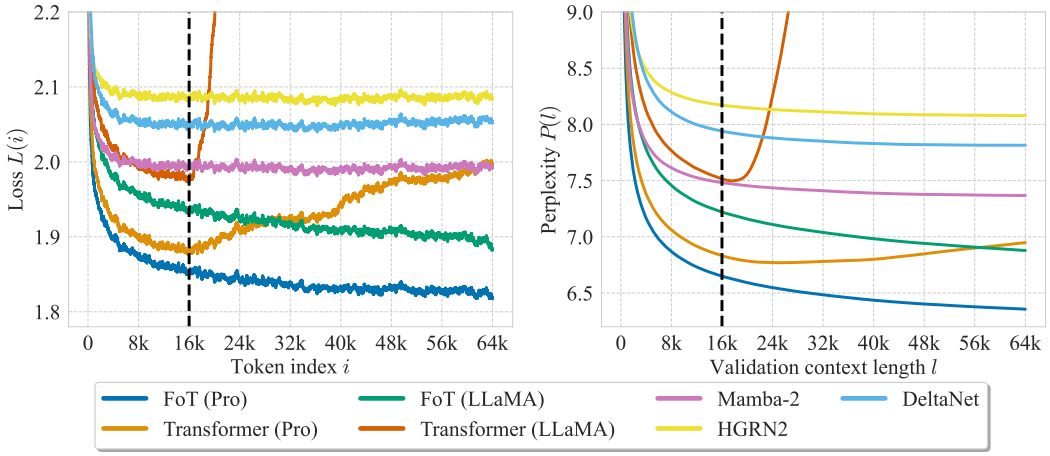

Figure 16: Results with 360M-parameter models trained on roughly 7.5B tokens. (**left**) Per-token loss $L(i)$ at different token position i. (**right**) Validation perplexity $P(l)$ over different validation context length l. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

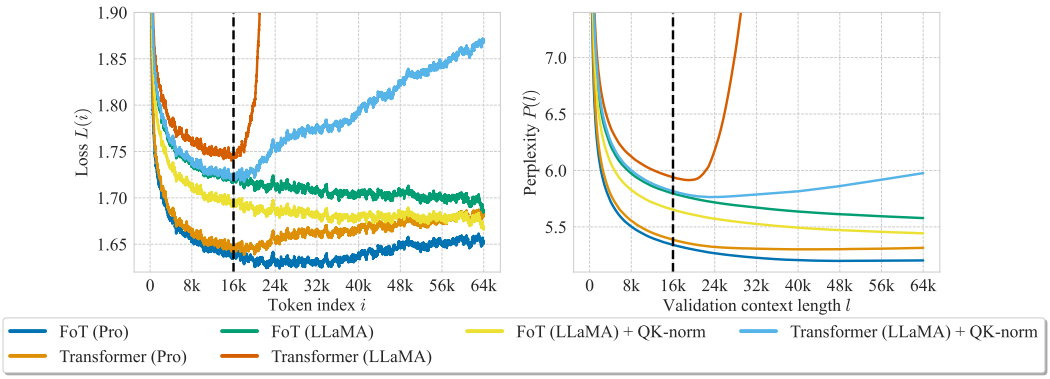

Figure 17: Additional results with FoT (LLaMA) + QK-norm and Transformer (LLaMA) + QK-norm. (**left**) Per-token loss $L(i)$ at different token position i. (**right**) Validation perplexity $P(l)$ over different validation context length l. All models have 760M parameters and are trained on roughly 16B tokens. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

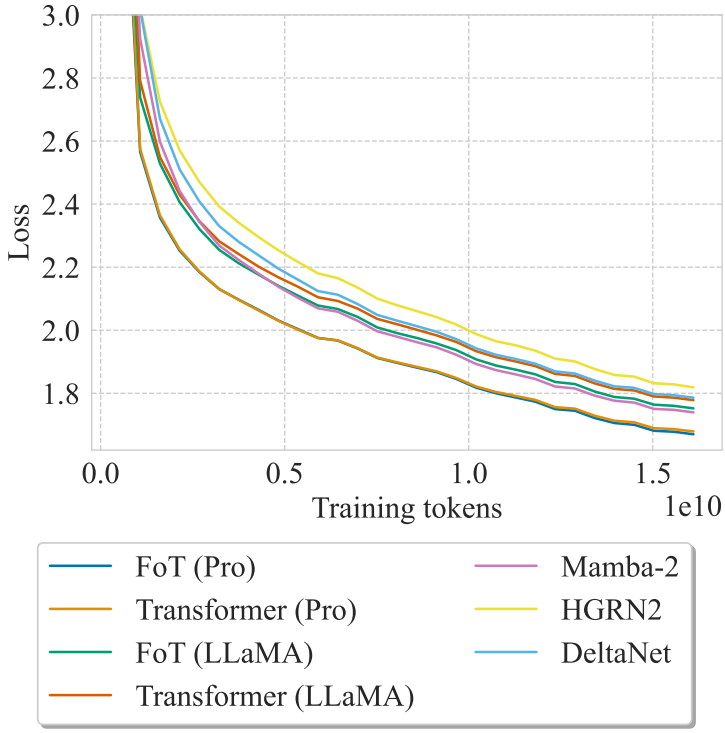

Figure 18: Training curves of different models presented in Figure 2. These curves show the training loss averaged every 512×2^{20} tokens. All models have 760M parameters.

E.9 Training curves

In Figure 18 we show the training curve for all models presented in Figure 2. Note the the performance gap between FoT (Pro) and Transformer (Pro) only start to appear near the end of training.

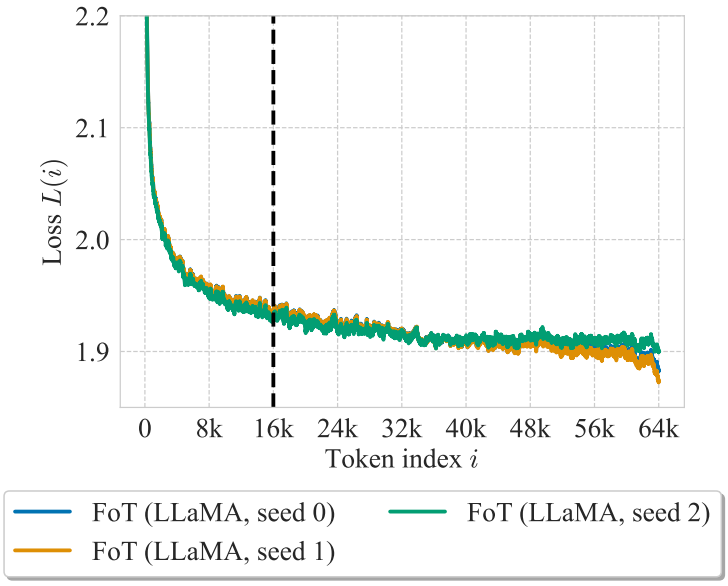

Figure 19: Result stability across seeds with 360M-parameter FoT (LLaMA). All models are trained on roughly 7.5B tokens. The vertical dashed line indicates the training context length. The per-token loss is typically noisy, so we smooth the curve using a moving average sliding window of 101 tokens. In this plot $1k = 1024$.

E.10 Stability across random seeds

Although it is very challenging for us to run multiple seeds for all our results (running n seeds is n times more expensive) due to limited computational resources, we have run three seeds for our 360M-parameter FoT (LLaMA) model to show that the variance across seeds is small. The results are shown in Figure 19. Note the per-token loss curves within the training context length largely overlap. The variance beyond the training context length is larger, but the overall loss trend of different runs remain qualitatively similar. This is expected since there is no constraint for model behavior beyond the training context length during training.

E.11 Impact of parameter initialization

We find the baseline Transformer to be sensitive to parameter initialization. In particular, we find the HuggingFace LLaMA initialization to perform significantly better than the a legacy initialization in the Flash Linear Attention repository (Yang & Zhang, 2024) for the Transformer. The former uses a simple normal distribution initialization with a standard deviation 0.02 for all linear layers while the latter uses a combination of Xavier uniform (Glorot & Bengio, 2010), normal distribution, and rescaling of MLP output projection layers based on the number of layers.[6] The impact of initialization for the Transformer and the Forgetting Transformer is shown in Figure 20.

E.12 Additional visualization of forget gate matrix

In Figure 21 we show the forget gate matrix $\boldsymbol{F}$ from 16 heads distributed in 4 layers. Note that since these matrices are large (16384×16384), if only near-diagonal entries are non-zero the visualization will look almost all black.

[6]See `https://github.com/sustcsonglin/flash-linear-attention/blob/0ce8ce336a8472346e5877b26151e982734c63bb/fla/layers/attn.py#L76-L83` and `https://github.com/sustcsonglin/flash-linear-attention/blob/0ce8ce336a8472346e5877b26151e982734c63bb/fla/models/transformer/modeling_transformer.py#L148-L179` for details

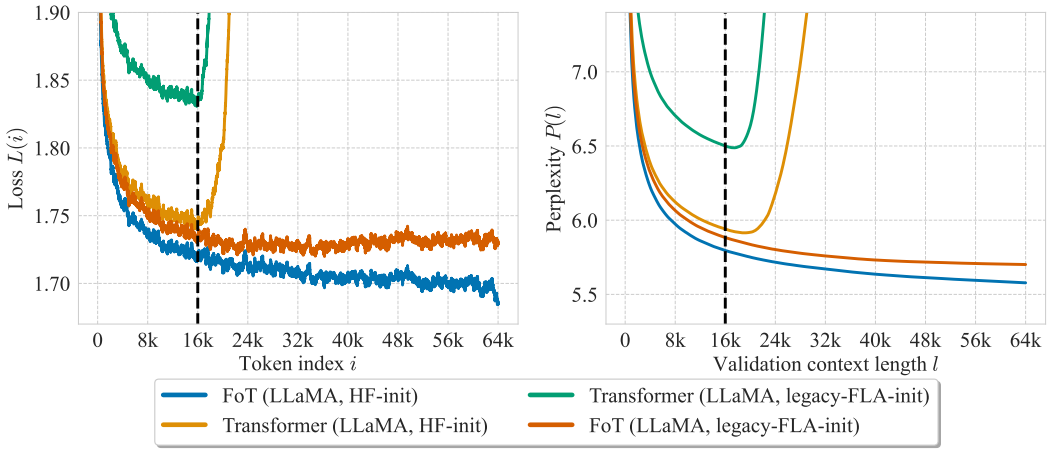

Figure 20: Effect of different parameter initialization. All models are roughly 760M parameters and are trained on roughly 16B tokens on LongCrawl64. The vertical line indicates the training context length.

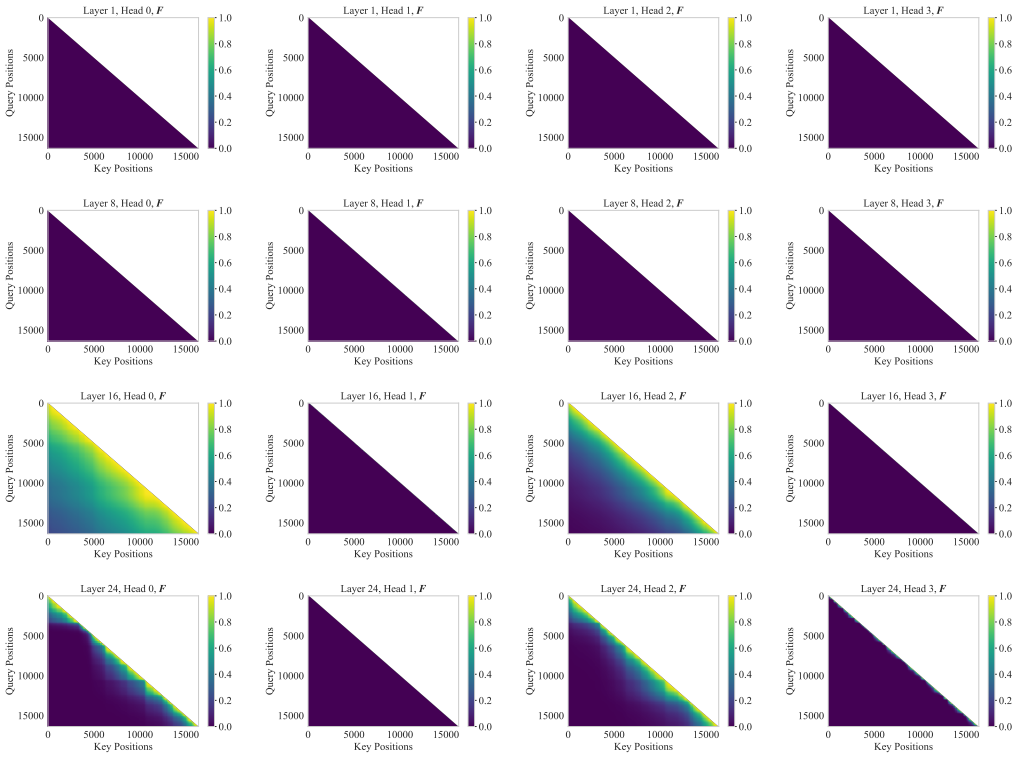

Figure 21: Visualization of the forget gate weight matrix $\boldsymbol{F}$ from 16 heads in 4 different layers. These results use FoT (LLaMA).