Decoder-Hybrid-Decoder Architecture for Efficient Reasoning with Long Generation

Anonymous Authors[1]

Abstract

Recent advances in language modeling have demonstrated the effectiveness of State Space Models (SSMs) for efficient sequence modeling. While hybrid architectures such as Samba and the decoder-decoder architecture, YOCO, have shown promising performance gains over Transformers, prior works have not investigated the efficiency potential of representation sharing between SSM layers. In this paper, we introduce the Gated Memory Unit (GMU), a simple yet effective mechanism for efficient memory sharing across layers, and apply it to create a decoder-hybrid-decoder architecture, SambaY, through integrating GMUs into the cross-decoder of YOCO. SambaY significantly enhances decoding efficiency, preserves linear pre-filling time complexity, and boosts long-context performance, all while eliminating the need for explicit positional encoding. Through extensive scaling experiments, we demonstrate that our architecture exhibits a significantly lower irreducible loss compared to a strong YOCO baseline, indicating superior performance scalability under large-scale compute regimes. Our largest model enhanced with Differential Attention, Phi4-mini-Flash-Reasoning, achieves comparable performance to Phi4-mini-Reasoning on reasoning tasks such as Math500, AIME24, and GPQA Diamond, while delivering up to 10× higher decoding throughput on 2K-length prompts with 32K generation length under the vLLM inference framework.

1. Introduction

State Space Models (SSMs) (Gu et al., 2021; 2022; Gu & Dao, 2023; Dao & Gu, 2024), including linear attention (Hua et al., 2022; Sun et al., 2023; Qin et al., 2022; Yang et al., 2024; 2025) and modern Recurrent Neural Networks (RNNs) (Beck et al., 2024; 2025; Peng et al., 2023; Goldstein et al., 2024) have recently shown promising results for more efficient sequence modeling over Transformers (Vaswani et al., 2017). While pure SSMs offer computational advantages with linear complexities, hybrid architectures (Lieber et al., 2024; De et al., 2024; Ren et al., 2025; Waleffe et al., 2024; Dong et al., 2025; MiniMax, 2025) combining SSMs with self-attention can bridge the expressiveness gap of SSMs/RNNs to Transformers with a few attention layers (Wen et al., 2024). Notably, the decoder-decoder architecture, YOCO (Sun et al., 2024), accelerates inference by storing the Key-Value (KV) pairs from just one attention layer and re-using them across all subsequent layers, a strategy that has delivered substantial pre-filling performance gains in practice. However, challenges remain; YOCO does not mitigate the attention memory I/O cost for its cross-attention layers during decoding. This limitation becomes particularly pronounced for modern large language models (LLMs) (OpenAI, 2024; DeepSeek-AI, 2025) that generate extensively long Chains-of-Thought (CoTs) (Wei et al., 2022) for hard reasoning tasks.

In this paper, we investigate the potential of representation sharing between SSM layers to enhance decoding efficiency. We introduce the Gated Memory Unit (GMU), a versatile, simple yet effective mechanism for efficient memory sharing across layers. Applying GMUs to the cross-decoder of YOCO, we create a novel decoder-hybrid-decoder architecture named SambaY that uses Samba (Ren et al., 2025) for the self-decoder and replaces half of the cross-attention layers with GMUs to share the inner representations of the final SSM layers in the self-decoder. Since around 50% of expensive cross-attention layers are replaced with cheap element-wise gating, SambaY significantly improves decoding efficiency and maintains a linear pre-filling time complexity, all while removing the need for explicit positional encoding such as RoPE (Su et al., 2021).

To enable a robust comparison of the scaling capabilities across different architectures, we first design a principled μP++ hyperparameter transfer scheme that accounts for both depth and width scaling, as well as the application of weight decay to vector-like parameters. We then conduct extensive

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

experiments up to 3.4B parameters/600B tokens to verify the scaling behaviors of both our μP++ scaling laws and the SambaY architecture. Comparing to Samba+YOCO, an architecture that naively combines Samba with YOCO, we show that SambaY has significantly lower irreducible loss (Hestness et al., 2017) on the validation set when scaling with the training FLOPs, indicating a better scaling potential with large-scale computes. We also conduct extensive experiments to verify the long-context retrieval capabilities of our architecture. Our results reveal that SambaY achieves superior performance on challenging long-context tasks like Phonebook and RULER (Hsieh et al., 2024) benchmark, even with a modest Sliding Window Attention (SWA) size of 256. To further explore the capabilities of hybrid models with a single set of full attention memory, we augment SambaY with Differential Attention (Ye et al., 2024), resulting in the Phi4-mini-Flash architecture. We pre-train our 3.8B-parameter model Phi4-mini-Flash with 5T tokens from the same Phi4-mini data corpus and further follow the recipe as Phi4-mini-Reasoning (Xu et al., 2025) to produce our reasoning model, Phi4-mini-Flash-Reasoning. Our model achieves performance comparable to the strong Phi4-mini-Reasoning baseline on challenging reasoning benchmarks such as Math500, AIME24 and GPQA Diamond. Critically, Phi4-mini-Flash-Reasoning delivers up to 10$\times$ higher decoding throughput on 2K-length prompts with 32K generation length under the vLLM (Kwon et al., 2023) inference framework, showcasing its substantial and practical efficiency gains for the LLM reasoning paradigm of generating long Chain-of-Thoughts.

2. Decoder-Hybrid-Decoder Architecture

Inspired by the gating mechanism that broadly exists in Gated Linear Units (Shazeer, 2020), Gated Attention Units (Hua et al., 2022) and SSMs (Gu & Dao, 2023; Yang et al., 2025), we first introduce our Gated Memory Unit (GMU) that takes the current layer's input representation and a previous layer's memory state as the inputs and outputs the gated representations with learnable projections. We then explore a specific application of GMUs to YOCO which produces our decoder-hybrid-decoder architecture. A dedicated related works section is included in Appendix G.

Gated Memory Unit (GMU). From an inter-layer perspective, we define "memory" as hidden representations passed from preceding layers. Specifically, at a given layer l, GMU operates on two inputs: the current layer's input hidden state, $\mathbf{x}_l \in \mathbb{R}^{d_m}$, and a memory state, $\mathbf{m}_{l'} \in \mathbb{R}^{d_h}$, from a previous layer l' (where $l' < l$). The GMU then produces an output $\mathbf{y}_l \in \mathbb{R}^{d_m}$ through a gating mechanism modulated by learnable projections. Formally, the GMU

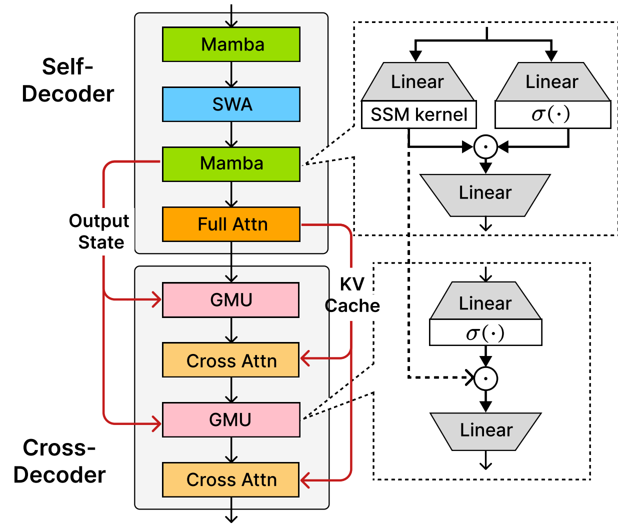


Figure 1: Our decoder-hybrid-decoder architecture taking Mamba as an exemplar SSM. Gated Memory Units (GMUs) are interleaved with the cross-attention layers in the cross-decoder to reduce the decoding computation complexity. Following YOCO (Sun et al., 2024), the full attention layer only calculates the KV cache during pre-filling, resulting a linear computation complexity.

can be expressed as:

$$\mathbf{y}_l = \mathbf{m}_{l'} \odot \sigma(W_1 \mathbf{x}_l) W_2$$

where $\sigma(\cdot)$ is the SiLU (Elfwing et al., 2017) activation function, $\odot$ denotes element-wise multiplication, and $W_1, W_2 \in \mathbb{R}^{d_h \times d_m}$ are learnable weight matrices. Intuitively, this gating mechanism allows the current layer's input $\mathbf{x}_l$ to selectively filter the information flowing from the memory $\mathbf{m}_{l'}$, effectively acting as a dynamic fine-grained recalibration of token mixing that occurred in previous layers based on the current query context for each of the memory channels. While in this work we primarily focus on gating memory from SSM layers (where d_h would correspond to the SSM inner dimension), the concept is generalizable. For instance, $\mathbf{m}_{l'}$ could be the intermediate output of a preceding attention layer, allowing GMUs to diversify the attention map for each channel of the value vectors based on the input representation of the current layer. Similarly, it could gate intermediate outputs from MLP layers, enabling retrieval from static, parametric memory. In both cases, GMUs save parameters and computation compared to the vanilla attention or the MLP layers.

Model architecture. In Figure 1, we illustrate our SambaY architecture, a decoder-hybrid-decoder architecture with Samba (Ren et al., 2025) as the self-decoder. We apply GMUs to the cross-decoder of YOCO to replace half of its cross-attention layers. The GMUs share the representation from the last SSMs layers in the self-decoder so that the pre-filling time complexity is still linear. Compared to YOCO, our approach only requires caching an additional SSM kernel output state $\mathbf{m} \in \mathbb{R}^{d_h}, d_h = 2d_m$ from the final Mamba layer—an overhead that is negligible in size—alongside the KV cache from the last full-attention layer during pre-filling. During decoding, we reduce the memory I/O complexity for half of the cross-attention layers from a linear cost of $O(d_{kv}N)$ to a constant $O(d_h)$, where N is the sequence length and d_{kv} is the key-value dimension. This leads to significant efficiency gains when $N \gg d_h/d_{kv}$, a condition that is easily met in practice since the ratio d_h/d_{kv} typically does not exceed 128.

3. Experiments & Results

Motivated by the theoretical efficiency of our SambaY architecture, we aim to address the following research questions: Does the architecture scale effectively? Does it compromise long-context performance? Can it support reasoning over long generations? Given that a neural architecture's performance is tightly coupled with its optimization and initialization settings, we begin by establishing a generic scaling setup to encourage a fair comparison of scaling behavior across different architectures.

3.1. Scaling Experiments on Open-Source Data

Architecture scaling setup. We use a simple linear rule from the previous works on Transformer models (Kaplan et al., 2020; Tian et al., 2024) for scaling the architectural shape of our Transformer++ baseline, including model width w, model depth d, number of attention query heads h_q and the MLP inner dimension w_{mlp}, *i.e.*,

$$w = \alpha d, \quad \alpha = \alpha_0 = 128,$$

$$h_q = d, \quad h_{kv} = d/4, \quad w_{mlp} = 4w,$$

where the Transformer-specific aspect ratio α_0 and the number of key-value heads h_{kv} are computed based on the Llama-3-8B architecture. We use SwiGLU (Shazeer, 2020) for MLP and RoPE (Su et al., 2021) with the base frequency of 10,000. The total number of non-embedding parameters for the Transformer++ architecture can then be calculated as,

$$N_{\text{attn}}(d) = 2.5dw^2, N_{\text{mlp}}(d) = 12dw^2,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mlp}}(d) = 14.5dw^2 = 237568d^3.$$

Baseline Architectures. We consider several architectural variants alongside our proposed SambaY architecture and Transformer++, including Samba+YOCO (which uses Samba as self-decoder for the original YOCO architecture), TransformerLS (interleaving SWA with full attention using a layer ratio of 3:1), and SambaY+DA (which uses Differential Attention (DA) (Ye et al., 2024) for all attention layers). All hybrid architectures maintain consistent hyperparameter settings with a 4× MLP inner dimension expansion ratio and GQA (Ainslie et al., 2023) group size of 4 for self-attention layers, matching our Transformer++ baseline. To ensure fair comparison, we standardize the sliding window size to 128 for all architectures with SWA while conducting extensive ablations on window size effects in Section 3.2. Following the studies in recent hybrid models (Lieber et al., 2024; Ren et al., 2025), we omit explicit positional encodings (NoPE) for all hybrid SSMs architectures. Detailed configurations for the implementation of DA are provided in Appendix E.

Scaling transfer for hybrid architectures. Since different token mixers have their own inner dimension expansion ratio, it is hard to balance the number of parameters between hybrid models and Transformers to make fair comparisons. Previous works (DeepSeek-AI, 2024a; Ren et al., 2025; Yang et al., 2025) often adjust the model depth to tie the total number of parameters, but this could change the memory cache size significantly (e.g. adding two attention layers in a 12-layer Transformer resulting in a 16.7% increase of KV cache size), making unfair comparisons regarding the inference time cost. We propose a simple solution that (1) builds an iso-parametric equation with respect to the aspect ratio via aligning the total number of non-embedding parameters to the Transformer baseline, (2) solves the equation to obtain the specific aspect ratio (which is rounded up to an even integer to guarantee the activation of Tensor Cores[1]) for the hybrid architectures. We also fix the head dimension to be $\alpha_0 = 128$, and set the inner dimension of the attention layers to be $w_{\text{attn}} = \alpha_0 d$ so that the number of key-value heads h_{kv} is a valid integer. Specifically, for SambaY, we have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/4,$$

$$N_{\text{mamba}}(d) = 6dw^2/4, \quad N_{\text{gmu}}(d) = 4dw^2/4,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d)$$
$$= 144\alpha d^3 + 14.5\alpha^2 d^3 = 237568d^3.$$

Solving for α, we get $\alpha_1 \approx 124$. For Samba+YOCO, we can similarly solve an iso-parametric equation to obtain $\alpha_2 \approx 126$, with more details in appendix A.

Hyperparameter scaling with μP++. To account for both width and depth scaling of model architectures, we propose μP++ hyperparameter scaling laws that integrate μP (Yang et al., 2022), Depth-μP (Yang et al., 2023), and apply

[1] https://developer.nvidia.com/blog/optimizing-gpu-performance-tensor-cores/

zero weight decay to vector-like or scalar-like parameters[2] for training stability. Since we use the AdamW optimizer (Loshchilov & Hutter, 2018), we apply batch-size-based scaling with $\eta \propto \sqrt{B}$. The learning rate is further scaled as $\eta \propto 1/\sqrt{d}$ following Depth-μP. For studying the FLOPs scaling behavior across model architectures, we adopt the Chinchilla scaling law (Hoffmann et al., 2022) to scale the number of training tokens T linearly with the number of model parameters. Formally, we have

$$\eta = \eta_0 \sqrt{\frac{Bd_0}{B_0 d}}, \quad B = B_0, \quad T = T_0 \frac{N(d)}{N(d_0)},$$

where the base learning rate is set as $\eta_0 = 4 \times 10^{-4}$ and the base batch size $B_0 = 2^{21}$ number of tokens. We also explore scaling the batch size sub-linearly with respect to the training tokens (McCandlish et al., 2018; Shuai et al., 2024; Li et al., 2025) (more details in Appendix B), but find that it harms the data scaling behavior of the models, so we keep the batch size as a constant across scales. The base model depth is set as $d_0 = 16$ so that $N(d_0) \approx 10^9$ number of parameters. The base training tokens T_0 is set to 100B. We adopt μP to scale the output logits and the learning rate of matrix-like parameters proportional to $1/w$, and the output projection of each layer is divided by $\sqrt{2d}$ following Depth-μP. The base attention logits multiplier is set to $1/\sqrt{\alpha}$. We fix other hyper-parameters of the optimizer with $\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 10^{-8}$ and a weight decay of 0.1. A linear learning rate schedule is applied with 1B warm-up tokens increasing to the peak learning rate η, followed by a linear decay to zero. We use LeCun uniform initialization (*i.e.* PyTorch default initialization) (LeCun et al., 2012) for the weight matrices following (Gu & Dao, 2023) and (Ren et al., 2025), and tie the input and output embedding matrices which are initialized from the normal distribution $\mathcal{N}(0, 0.02^2)$.

Scaling experiment setups. A common concern with SSMs is that they are not theoretically more expressive than self-attention for in-context retrieval (Wen et al., 2024). This raises the question of whether the better performance of hybrid SSM models is owing to their fast convergence from the recency bias, while Transformers could eventually match their performance given more training tokens. With the scaling laws we established in the previous paragraphs, we can now examine this hypothesis systematically. We first study the data scaling behavior across architectures. Specifically, we fix the model size at around 1B parameters with the architecture parameterization of $d = 16$ and scale the number of training tokens T from 100B to 600B. We also study the FLOPs scaling behaviors of the model architectures with up to 3.4B parameters and 342B tokens through varying the model depth $d = \{8, 12, 16, 20, 24\}$. We use a 4K training sequence length and the SlimPajama (Soboleva et al., 2023) dataset for all our scaling experiments and measure the model performances on its validation set.

[2] Following the definition in μP, parameters are vector-like when exactly one dimension scales with model width (e.g., embedding and unembedding layers), and scalar-like when no dimension scales with width.

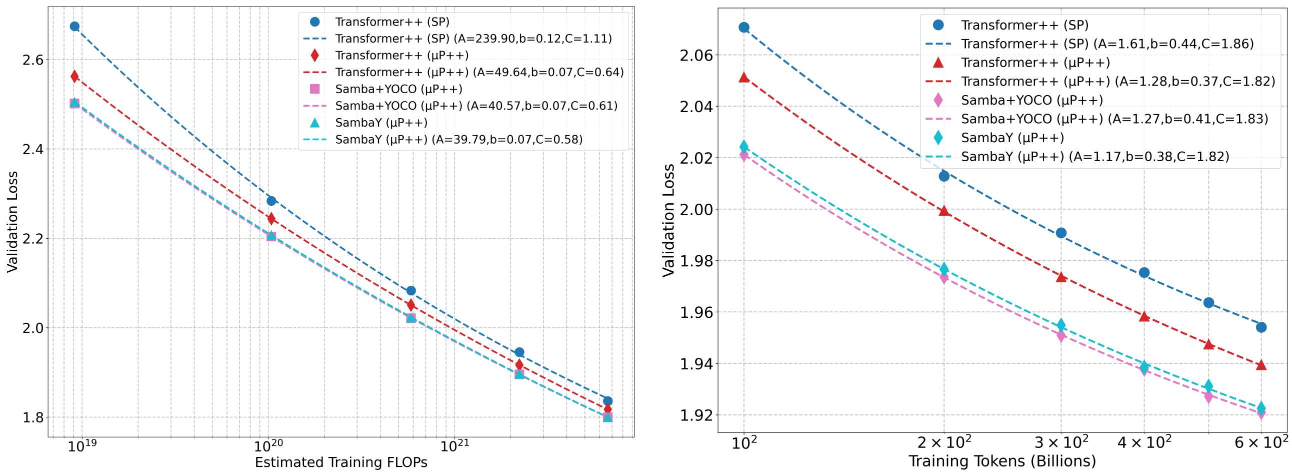
Comparison of scaling behaviors. To quantitatively compare the scaling trajectories, we fit the validation loss L as a function of compute (FLOPs), denoted as D_{FLOPs}, to a power law (Hestness et al., 2017; Hoffmann et al., 2022) of the form:

$$L(D_{\text{FLOPs}}) = A \cdot D_{\text{FLOPs}}^{-b} + C$$

This model enables us to estimate the irreducible loss C which represents the lower bound of achievable loss for a given architecture or parameterization under infinite compute, and the scaling exponent b that reflects the learning efficiency with respect to compute. We fit the curves with least squares and the LMA algorithm (LEVENBERG, 1944; Marquardt, 1963). A similar power law model is employed for data scaling experiments, where loss is modeled as a function of the number of training tokens D_{tokens}.

In Figure 2, we present the results of both FLOPs scaling and data scaling experiments, showing validation loss on the SlimPajama dataset as a function of total training FLOPs or number of training tokens. We show both the original data points and the fitted power-law curves. The goodness of fit for each curve is assessed using the R^2 statistic and we observe that all plots have a fitness score $R^2 \geq 0.999$, indicating near-perfect fits. While larger values of the scaling exponent b or the coefficient A indicate that a model may converge more rapidly given a small-scale compute or data budget, these parameters alone do not necessarily predict superior performance at larger scales. Therefore, we primarily emphasize the irreducible loss C obtained from scaling law fitting as the principal metric for assessing an architecture's long-term scaling potential. As illustrated in Figure 2a, the SambaY architecture exhibits the lowest irreducible loss ($C = 0.58$) for FLOPs scaling, suggesting that it can attain a superior validation loss compared to other architectures when scaled further with substantially increased computational resources. We also observe that μP++ yields a lower irreducible loss than Standard Parameterization (SP) under both data and compute scaling, indicating more favorable scaling potentials. More experimental details are included in Appendix C.

Notably in Figure 2b, the Transformer++ model trained with μP++ exhibits a substantial validation loss gap compared to SambaY and SambaY+YOCO within the measured range of training tokens. However, its fitted irreducible loss ($C = 1.82$) is nearly identical to those of the hybrid models, suggesting that with an infinite amount of data, Transformer++ can eventually catch up—albeit with slower



(a) Compute scaling comparisons

(b) Data scaling comparisons

Figure 2: Validation Loss v.s. FLOPs (left) or Training Tokens (right) on the SlimPajama dataset. Besides the architecture comparisons, we also compare our μP++ based scaling with the Standard Parametrization (SP).

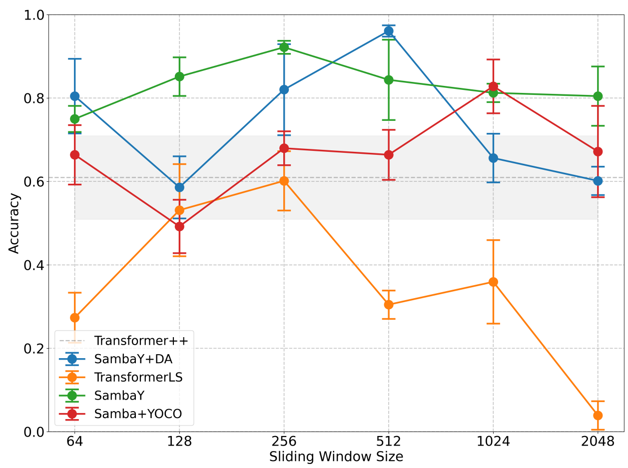


Figure 3: Accuracy (with error bars) v.s. Sliding Window Size on Phonebook with 32K evaluation length.

convergence. This aligns with our expectation, as there is no theoretical expressiveness gap between Transformers and our hybrid models when the number of parameters is held constant. Interestingly, this convergence no longer holds when both model size and data scale proportionally. As illustrated in Figure 2a, under the μP++ setting, Transformer++ exhibits the highest irreducible loss $C = 0.64$, indicating that hybrid architectures could offer superior scalability under limited data regimes.

3.2. Efficient Long Context Retrieval

Given the presence of full-attention layers, we aim to determine the minimal size of the sliding window attention required for our hybrid models to retain effective long-context retrieval capabilities. Specifically, we pre-train 1.0B parameter models with μP++ and $d = 16$ using TransformerLS, SambaY, SambaY+DA and Samba+YOCO architectures respectively on the ProLong-64k (Gao et al., 2024) dataset with 32K sequence length and 40B tokens, varying the window size of their Sliding Window Attention (SWA) in the range $\{64, 128, \ldots, 2048\}$. We align the number of parameters between different architectures through building the iso-parametric equations as in Section 3.1. We adopt variable-length training, where short documents are packed together and self-attended within the same segment. We evaluate the long-context retrieval capabilities of the models using a difficult Phonebook benchmark (Jelassi et al., 2024) with a 32K context length (containing 1,850 name-number pairs). We choose this benchmark because it is a realistic multi-key-value retrieval task with minimal instructions, which minimizes the confounding influence of instruction-following ability when evaluating long-context retrieval performance. We use a RoPE base of 640K for TransformerLS and Transformer++, following the lower bounds proposed in (Xu et al., 2024). We also examine how the training corpus and methods affect the long context performance of these models, with more details in Appendix D.

As shown in Figure 3, surprisingly, larger SWA sizes do not consistently provide better results. We speculate that learned full attention involves both sliding window (local) patterns and non-local patterns like global retrieval or attention sinks. Using small to intermediate sliding window sizes, where models like SambaY and SambaY+DA show strong performance, could enable the model to focus on local patterns more easily and possibly mitigate issues like attention sinks (Gu et al., 2025). Moreover, shorter sliding windows might facilitate faster convergence, a crucial factor

Table 1: Retrieval accuracy on Needle-In-A-Haystack (NIAH) tasks with 32K context from the RULER (Hsieh et al., 2024) long context benchmark. MK: Multi-Key, MQ: Multi-Query, MV: Mutli-Value, S: Single-needle. We use the best Sliding Window Attention (SWA) size found on the Phonebook benchmark for each hybrid architecture. Best results are in bold, second best underlined.

Model	SWA	MK-1	MK-2	MK-3	MQ	MV	S-1	S-2	S-3	Avg.
Transformer++	-	36.4	3.8	0.0	<u>27.9</u>	<u>24.1</u>	94.8	66.0	31.0	35.5
TransformerLS	256	42.8	6.0	0.0	**29.8**	**27.5**	91.8	49.6	23.4	33.9
Samba+YOCO	1024	49.0	**28.0**	**2.6**	12.8	18.3	**100.0**	63.2	23.6	37.2
SambaY	256	<u>54.6</u>	<u>27.8</u>	<u>0.4</u>	12.7	19.4	83.2	<u>81.2</u>	<u>63.8</u>	<u>42.9</u>
SambaY+DA	512	**64.6**	27.6	0.2	12.8	19.9	<u>99.8</u>	**86.4**	**69.6**	**47.6**

in long context training scenarios often characterized by limited high-quality data. The lower scores of TransformerLS (orange line), which consistently underperforms the SambaY variants and reaches a peak accuracy of only 0.602 at an SWA of 256, could be indicative of Transformer-based models requiring more substantial data for long-context training.

Using the optimal sliding window size from the Phonebook benchmark, we evaluate our architectures on both long-context retrieval tasks (Table 1) and traditional downstream benchmarks (Table 2). Across both contexts, hybrid models with SSMs consistently outperform pure Transformer architectures. SambaY variants demonstrate notable advantages in long-context retrieval while maintaining strong performance on short-context tasks, despite using much smaller sliding window sizes than Samba+YOCO. The addition of DA further enhances multi-key and single-needle retrieval capabilities, while TransformerLS shows specific strengths in multi-query and multi-value scenarios. Overall, these results suggest that GMUs facilitate efficient information sharing across layers, enabling strong performance with smaller SWA sizes and offering better balance between computational efficiency and model capability.

3.3. Large-Scale Pre-training on High-quality Proprietary Data

We apply our hybrid model architecture to pre-train a larger-scale prototype model named Phi4-mini-Flash. It incorporates the SambaY architecture alongside Differential Attention (DA) (Ye et al., 2024) with an SWA size of 512 and GQA of group size 2. Compared to the configuration described in Section 3.1, this model uses a different aspect ratio $\alpha = 80$ and an attention head dimension of 64. It is trained with standard parameterization rather than μP++ due to resource constraints during the scaling study. We pretrain our model on 5T tokens from the data corpus used by Phi4-mini (Microsoft et al., 2025) on 1K A100-80GB GPUs for 14 days. During training, we encounter severe loss divergence, which we mitigate by introducing label smoothing of 0.1 and attention dropout of 0.05. The optimization setup here is by no means optimal, as the primary goal of this experiment is to evaluate the viability of our architecture at larger scales. Phi4-mini-Flash uses a 200K token vocabulary size consistent with Phi4-mini. Additional training and architectural details are provided in Appendix E.

Table 3 demonstrates that Phi4-mini-Flash outperforms the Phi4-mini baseline across a diverse range of tasks, with notable improvements on knowledge-intensive benchmarks like MMLU (4.6% absolute gain) and coding tasks such as MBPP (4.5% absolute gain). The consistent performance advantage, winning on 7 out of 8 benchmarks, is particularly significant given that Phi4-mini-Flash achieves these gains while maintaining substantially higher computational efficiency during inference.

3.4. Efficient Reasoning with Long Generation

Our Phi4-mini-Flash-Reasoning model is continually trained from the Phi4-mini-Flash model with the multi-stage distillation recipe following Phi4-mini-Reasoning (Xu et al., 2025). As shown in Table 4 and Figure 4, our reasoning model achieves performance comparable to Phi4-mini-Reasoning after SFT on AIME24 (Art of Problem Solving), Math500 (Hendrycks et al., 2021b), and GPQA Diamond (Rein et al., 2023), while employing a significantly more computationally efficient architecture, achieving up to 10× higher throughput in long-generation scenarios and 4.9× speedup in long-context processing. We evaluate the throughput of our vLLM implementation[3] using random weights to eliminate the influence of potentially shorter generation lengths on speed measurements, ensuring a fair comparison across different architectures. We use the same hyperparameter configurations as Phi4-mini-Flash for the YOCO and SambaY based baseline architectures. Notably, our DA implementation relies on a naive four-pass FlashAt-

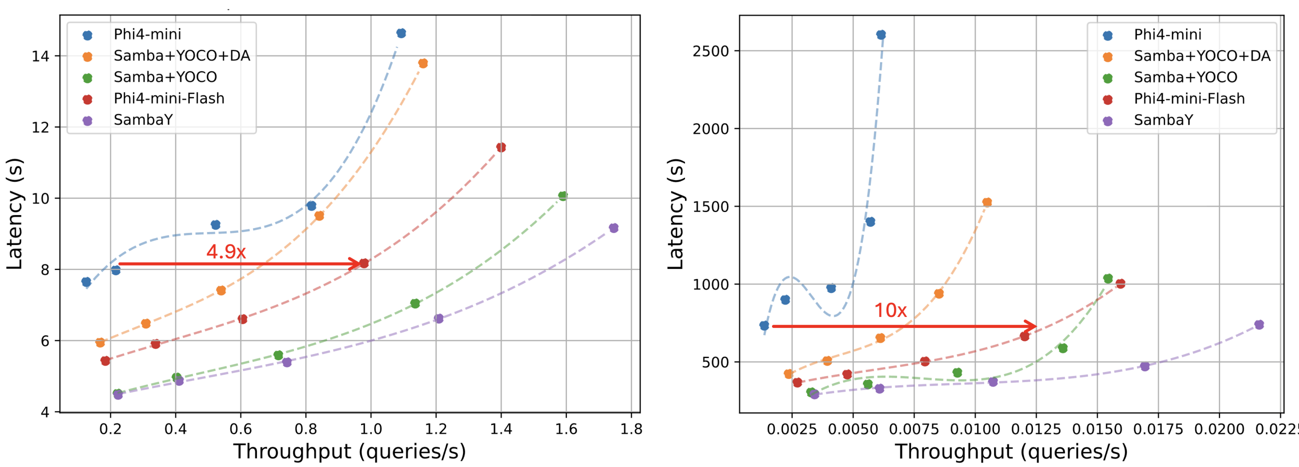
[3]We customize the official vLLM framework with the version 0.7.3 to support our Phi4-mini-Flash architecture.

Table 2: Downstream short-context evaluation on language modeling and common-sense reasoning tasks in zero-shot for 1B-parameter models with the tuned sliding window size. The training speed is measured in MTPS (Million Tokens Per Second) with 64 A100-80GB GPUs. Best results are in bold, second best underlined.

Model	**SWA**	**Speed** mtps ↑	**Wiki.** ppl ↓	**LMB.** acc ↑	**ARC-c** acc_n ↑	**ARC-e** acc ↑	**Hella.** acc_n ↑	**PIQA** acc ↑	**Wino.** acc ↑	**Avg.** acc ↑
Transformer++	-	0.89	19.75	45.45	27.56	54.17	43.86	68.77	50.28	48.35
TransformerLS	256	**1.46**	18.49	48.77	28.84	57.11	45.85	69.21	53.67	50.57
Samba+YOCO	1024	0.99	16.73	**50.53**	28.50	60.02	48.85	**71.55**	52.57	52.00
SambaY	256	1.11	17.83	50.40	**29.44**	57.87	49.08	71.00	**55.25**	**52.17**
SambaY+DA	512	0.91	**16.59**	49.68	28.33	**60.65**	**49.53**	71.38	53.43	**52.17**

Table 3: Downstream evaluation performance of post-trained models. We use the completion split for BigCodeBench evaluation. Bold indicates the best result per row.

Benchmark	**Metric**	**Phi4-mini**	**Phi4-mini-Flash**
MMLU (Hendrycks et al., 2021a)	5-shot	67.3	**71.9**
MMLU-Pro (Wang et al., 2024)	0-shot, CoT	52.8	**54.7**
Arena Hard (Li et al., 2024)	Win Rate	32.8	**34.9**
GSM8K (Cobbe et al., 2021)	0-shot, CoT	88.6	**89.5**
Qasper (Dasigi et al., 2021)	F1	**40.4**	40.2
SummScreenFD (Chen et al., 2022)	ROUGE-L	16.0	**17.0**
BigCodeBench (Zhuo et al., 2025)	pass@1	43.0	**44.5**
MBPP (Austin et al., 2021)	pass@1	65.3	**69.8**



(a) Prompt: 32000, Generation: 500

(b) Prompt: 2000, Generation: 32000

Figure 4: Throughput and latency of text generation with various architectures under the vLLM inference framework (using one A100-80GB GPU and no Tensor Parallelism). A normal distribution with 30% variance was applied to prompt and generation lengths with averages of 32000/2000 and 500/32000 respectively, following the setting in (Holmes et al., 2024).

tention setup, rather than the optimized custom kernel proposed in the original DA paper, leaving significant room for further speed optimization to catch up the efficiency of SambaY. We provide case studies on the generalization of our model's reasoning ability beyond mathematical problems in Appendix F.

4. Ablation Study

To systematically evaluate the design choices in our decoder-hybrid-decoder architecture, we conduct comprehensive ablation experiments. All ablation models are trained with 1.0B parameters on the ProLong-64K dataset with 40B tokens and 32K sequence length with variable length training,

Table 4: Pass@1 performance of models on reasoning benchmarks measured with a maximum generation length of 32K. We use the multi-stage distillation with Supervised Fine-Tuning (SFT), following the recipe in Phi4-mini-Reasoning.

Model	**AIME24**	**Math500**	**GPQA Diamond**
Phi4-mini	10.0	71.8	36.9
Phi4-mini-Reasoning (SFT)	50.0	90.4	48.3
Phi4-mini-Flash-Reasoning (SFT)	46.7	90.4	48.5

Table 5: Downstream evaluation on Phonebook 32K (PB-32k), language modeling and common-sense reasoning tasks in zero-shot for 1B-parameter models with a sliding window size of 128. We measure the training speed in MTPS (Million Tokens Per Second) with 64 A100-80GB GPUs. The average accuracy does not include results from the PB-32K. Best results in bold, second best underlined.

Model	**Speed** mtps ↑	**Wiki.** ppl ↓	**PB-32K** acc ↑	**LMB.** acc ↑	**ARC-c** acc_n ↑	**ARC-e** acc ↑	**Hella.** acc_n ↑	**PIQA** acc ↑	**Wino.** acc ↑	**Avg.** acc ↑
SambaY	1.10	**16.89**	**76.56**	<u>50.22</u>	28.58	59.18	**49.07**	70.84	**55.09**	52.16
SambaY-2	**1.40**	17.76	21.88	49.49	29.69	<u>59.68</u>	<u>48.71</u>	71.22	52.17	51.83
MambaY	0.94	17.29	12.50	**50.24**	28.84	59.64	48.27	<u>71.44</u>	52.80	51.87
MambaY-2	<u>1.35</u>	<u>16.99</u>	17.19	49.76	27.39	58.46	48.43	70.24	50.28	50.76
SambaY-A	1.11	18.12	58.59	49.85	<u>30.29</u>	59.60	48.41	71.33	54.06	<u>52.26</u>
SambaY-AA	1.25	17.03	46.88	49.93	28.50	59.05	48.69	**72.25**	53.91	52.06
SambaY-MLP	1.15	18.70	<u>64.84</u>	50.16	**30.38**	**60.69**	48.46	<u>71.44</u>	<u>54.78</u>	**52.65**

using a consistent SWA size of 128 as in the scaling experiments. We leverage μP++ with depth $d = 16$ and construct iso-parameter equations to maintain parameter count equivalence across all variants. We examine several architectural modifications of SambaY: (1) SambaY-2, which substitutes Mamba with Mamba-2 in the self-decoder; (2) MambaY, which employs only Mamba in the self-decoder without SWA layers; (3) MambaY-2, which uses only Mamba-2 in the self-decoder; (4) SambaY-A, which applies GMUs to gate intermediate representations from the last full attention layer in the self-decoder rather than from Mamba; (5) SambaY-AA, which entirely removes cross-attention in the cross-decoder and instead uses GMU to gate the intermediate representations from the middle full attention layer; and (6) SambaY-MLP, which uses GMUs to gate the intermediate representation from the linear projection branch of the SwiGLU right following the full attention layer. We aim to answer the following research questions given the ablation results in Table 5.

Alternative architectures for self-decoder in SambaY? We observe significant performance variations when testing alternative architectures for the self-decoder. While SambaY achieves impressive accuracy on the Phonebook benchmark, substituting Mamba with Mamba-2 in SambaY-2 causes a dramatic drop in performance. Similarly, MambaY, which employs only Mamba without SWA layers, performs poorly on long-context retrieval. MambaY-2 shows modest improvement over MambaY but still significantly underperforms SambaY. We suspect this is due to Mamba-2's coarse, scalar-valued forget gates, which may impair the self-decoder's ability to represent precise positional information. Additionally, the poor performance of MambaY highlights the critical role of SWA in enabling effective long-context modeling, as the recency bias alone appears insufficient for learning effective representations in self-decoder for the cross-decoder to complete complex retrieval tasks.

Is GMU effective for other memories besides SSMs? To investigate whether GMUs can effectively gate representations from sources other than SSMs, we examine SambaY-A and SambaY-AA, which gate attention output representations, and SambaY-MLP, which gates MLP intermediate representations. As shown in Table 5, these variants achieve respectable performance on downstream tasks, with SambaY-MLP even surpassing the original SambaY on average accuracy for short-context tasks. However, for the long-context task, PB-32K, we observe a clear hierarchy: SambaY > SambaY-MLP > SambaY-A > SambaY-AA. This pattern indicates that GMUs remain effective with alternative memory sources, but their performance on retrieval tasks depends significantly on the memory source's inherent characteristics. Gating attention/MLP representations performs worse than the original SambaY on Phonebook because they lack the recency bias that SSMs naturally provide, which is particularly beneficial for retrieving complicated information. SambaY-AA, which completely removes cross-attention, shows further degradation, highlighting the complementary value of having both cross-attention and GMU-gated memories.

5. Conclusion

In this work, we introduced the Gated Memory Unit (GMU), a simple yet effective mechanism for efficient memory sharing across layers in sequence models. By integrating GMUs into a decoder-hybrid-decoder architecture, SambaY, we achieved significant improvements in both computational efficiency and model performance. Our extensive scaling experiments demonstrated that SambaY exhibits a lower irreducible loss compared to strong baselines, indicating superior scaling properties with increasing computational resources. Our largest model, Phi4-mini-Flash-Reasoning, matched the performance of Phi4-mini-Reasoning on challenging math reasoning benchmarks while delivering substantially higher decoding throughput on long-context generations. Given that our architecture still retains a full attention layer with linear decoding complexity, future work could explore dynamic sparse attention mechanisms to further improve efficiency in extremely long sequence generation, particularly in agentic application scenarios. Additionally, adaptive selection of memory sharing strategies based on task characteristics and computational constraints presents a promising direction for enhancing flexibility and performance.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebr'on, F., and Sanghai, S. K. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.48550/arXiv.2305.13245. URL https://arxiv.org/abs/2305.13245v3.

Alabdulmohsin, I. M., Neyshabur, B., and Zhai, X. Revisiting neural scaling laws in language and vision. *Advances in Neural Information Processing Systems*, 35:22300–22312, 2022.

Art of Problem Solving. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2025-04-20.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models. *arXiv preprint arXiv: 2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732v1.

Bahri, Y., Dyer, E., Kaplan, J., Lee, J., and Sharma, U. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27):e2311878121, 2024.

Beck, M., Poppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M. K., Klambauer, G., Brandstetter, J., and Hochreiter, S. xlstm: Extended long short-term memory. *Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.04517v1.

Beck, M., Pöppel, K., Lippe, P., Kurle, R., Blies, P. M., Klambauer, G., Böck, S., and Hochreiter, S. xlstm 7b: A recurrent llm for fast and efficient inference. *arXiv preprint arXiv: 2503.13427*, 2025. URL https://arxiv.org/abs/2503.13427.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6239. URL https://doi.org/10.1609/aaai.v34i05.6239.

Bjorck, J., Benhaim, A., Chaudhary, V., Wei, F., and Song, X. Scaling optimal LR across token horizons. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=WYL4eFLcxG.

Brandon, W., Mishra, M., Nrusimha, A., Panda, R., and Kelly, J. R. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv: 2405.12981*, 2024. URL https://arxiv.org/abs/2405.12981v1.

Chen, M., Chu, Z., Wiseman, S., and Gimpel, K. SummScreen: A dataset for abstractive screenplay summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8602–8615, Dublin, Ireland, May 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.acl-long.589.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv: 1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457v1.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv: 2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168v2.

Dao, T. and Gu, A. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ztn8FCR1td.

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365.

De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S., Desjardins, G., Doucet, A., Budden, D., Teh, Y. W., Pascanu, R., Freitas, N. D., and Gulcehre, C. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv: 2402.19427*, 2024. URL https://arxiv.org/abs/2402.19427v1.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv: 2405.04434*, 2024a. URL https://arxiv.org/abs/2405.04434v1.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv: 2412.19437*, 2024b. URL https://arxiv.org/abs/2412.19437v1.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv: 2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948v1.

Dong, X., Fu, Y., Diao, S., Byeon, W., Chen, Z., Mahabaleshwarkar, A., Liu, S.-Y., Keirsbilck, M. V., Chen, M.-H., Suhara, Y., Lin, Y. C., Kautz, J., and Molchanov, P. Hymba: A hybrid-head architecture for small language models. *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2411.13676v1.

Duanmu, H., Yuan, Z., Li, X., Duan, J., Zhang, X., and Lin, D. Skvq: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*, 2024.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 2017. doi: 10.1016/j.neunet.2017.12.012.

Gao, T., Wettig, A., Yen, H., and Chen, D. How to train long-context language models (effectively). *arXiv preprint arXiv: 2410.02660*, 2024. URL https://arxiv.org/abs/2410.02660v1.

Goldstein, D., Obeid, F., Alcaide, E., Song, G., and Cheah, E. Goldfinch: High performance rwkv/transformer hybrid with linear pre-fill and extreme kv-cache compression. *arXiv preprint arXiv: 2407.12077*, 2024. URL https://arxiv.org/abs/2407.12077v1.

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 R'e, C. Efficiently modeling long sequences with structured state spaces. *International Conference On Learning Representations*, 2021.

Gu, A., Gupta, A., Goel, K., and Ré, C. On the parameterization and initialization of diagonal state space models. *ARXIV.ORG*, 2022. doi: 10.48550/arXiv.2206.11893.

Gu, X., Pang, T., Du, C., Liu, Q., Zhang, F., Du, C., Wang, Y., and Lin, M. When attention sink emerges in language models: An empirical view. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=78Nn4QJTEN.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv: 1712.00409*, 2017. URL https://arxiv.org/abs/1712.00409.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models. *ARXIV.ORG*, 2022. doi: 10.48550/arXiv.2203.15556. URL https://arxiv.org/abs/2203.15556v1.

Holmes, C., Tanaka, M., Wyatt, M., Awan, A. A., Rasley, J., Rajbhandari, S., Aminabadi, R. Y., Qin, H., Bakhtiari, A., Kurilenko, L., and He, Y. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv: 2401.08671*, 2024. URL https://arxiv.org/abs/2401.08671.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., and Ginsburg, B. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv: 2404.06654*, 2024. URL https://arxiv.org/abs/2404.06654v1.

Hu, S., Tu, Y., Han, X., He, C., Cui, G., Long, X., Zheng, Z., Fang, Y., Huang, Y., Zhao, W., Zhang, X., Thai, Z. L., Zhang, K., Wang, C., Yao, Y., Zhao, C., Zhou, J., Cai, J., Zhai, Z., Ding, N., Jia, C., Zeng, G., Li, D., Liu, Z., and Sun, M. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv: 2404.06395*, 2024. URL https://arxiv.org/abs/2404.06395v1.

Hua, W., Dai, Z., Liu, H., and Le, Q. V. Transformer quality in linear time. *International Conference On Machine Learning*, 2022.

Jelassi, S., Brandfonbrener, D., Kakade, S., and Malach, E. Repeat after me: Transformers are better than state space models at copying. *International Conference on Machine Learning*, 2024. doi: 10.48550/arXiv.2402.01032. URL https://arxiv.org/abs/2402.01032v1.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv: 2001.08361*, 2020. URL https://arxiv.org/abs/2001.08361v1.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. *Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180v1.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. *Efficient BackProp*, pp. 9–48. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-35289-8_3. URL http://link.springer.com/content/pdf/10.1007/978-3-642-35289-8_3.

Lee, W., Lee, J., Seo, J., and Sim, J. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/lee.

LEVENBERG, K. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2 (2):164–168, 1944. ISSN 0033569X, 15524485. URL http://www.jstor.org/stable/43633451.

Li, H., Zheng, W., Hu, J., Wang, Q., Zhang, H., Wang, Z., Xuyang, S., Fan, Y., Zhou, S., Zhang, X., and Jiang, D. Predictable scale: Part i - optimal hyperparameter scaling law in large language model pretraining. *arXiv preprint arXiv: 2503.04715*, 2025. URL https://arxiv.org/abs/2503.04715.

Li, T., Chiang, W.-L., Frick, E., Dunlap, L., Wu, T., Zhu, B., Gonzalez, J. E., and Stoica, I. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv: 2406.11939*, 2024. URL https://arxiv.org/abs/2406.11939.

Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., Abend, O., Alon, R., Asida, T., Bergman, A., Glozman, R., Gokhman, M., Manevich, A., Ratner, N., Rozen, N., Shwartz, E., Zusman, M., and Shoham, Y. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv: 2403.19887*, 2024. URL https://arxiv.org/abs/2403.19887v1.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Malladi, S., Lyu, K., Panigrahi, A., and Arora, S. On the SDEs and scaling rules for adaptive gradient algorithms. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=F2mhzjHkQP.

Marquardt, D. W. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963. ISSN 03684245. URL http://www.jstor.org/stable/2098941.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. An empirical model of large-batch training. *arXiv preprint arXiv: 1812.06162*, 2018. URL https://arxiv.org/abs/1812.06162v1.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Microsoft, Abouelenin, A., Ashfaq, A., Atkinson, A., Awadalla, H., Bach, N., Bao, J., Benhaim, A., Cai, M., Chaudhary, V., Chen, C., Chen, D., Chen, D., Chen, J., Chen, W., Chen, Y.-C., ling Chen, Y., Dai, Q., Dai, X., Fan, R., Gao, M., Gao, M., Garg, A., Goswami, A., Hao, J., Hendy, A., Hu, Y., Jin, X., Khademi, M., Kim, D., Kim, Y. J., Lee, G., Li, J., Li, Y., Liang, C., Lin, X., Lin, Z., Liu, M., Liu, Y., Lopez, G., Luo, C., Madan, P., Mazalov, V., Mitra, A., Mousavi, A., Nguyen, A., Pan, J., Perez-Becker, D., Platin, J., Portet, T., Qiu, K., Ren, B., Ren, L., Roy, S., Shang, N., Shen, Y., Singhal, S., Som, S., Song, X., Sych, T., Vaddamanu, P., Wang, S., Wang, Y., Wang, Z., Wu, H., Xu, H., Xu, W., Yang, Y., Yang, Z., Yu, D., Zabir, I., Zhang, J., Zhang, L. L., Zhang, Y., and Zhou, X. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras, 2025. URL https://arxiv.org/abs/2503.01743.

MiniMax. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv: 2501.08313*, 2025. URL https://arxiv.org/abs/2501.08313v1.

OpenAI. Openai o1 system card. *arXiv preprint arXiv: 2412.16720*, 2024. URL https://arxiv.org/abs/2412.16720.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *Annual Meeting of the Association for Computational Linguistics*, 2016. doi: 10.18653/v1/P16-1144. URL https://arxiv.org/abs/1606.06031v1.

Peng, B., Alcaide, E., Anthony, Q. G., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., Kranthikiran, G., He, X., Hou, H., Kazienko, P., Kocoń, J., Kong, J., Koptyra, B., Lau, H., Mantri, K. S. I., Mom, F., Saito, A., Tang, X., Wang, B., Wind, J. S., Wozniak, S., Zhang, R., Zhang, Z., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R. Rwkv: Reinventing rnns for the transformer era. *Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.48550/arXiv.2305.13048. URL https://arxiv.org/abs/2305.13048v1.

Qin, Z., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., and Zhong, Y. The devil in linear transformer. *Conference on Empirical Methods in Natural Language Processing*, 2022. doi: 10.48550/arXiv.2210.10340. URL https://arxiv.org/abs/2210.10340v1.

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

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv: 2311.12022*, 2023. URL https://arxiv.org/abs/2311.12022v1.

Ren, L., Liu, Y., Lu, Y., Shen, Y., Liang, C., and Chen, W. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bIlnpVM4bc.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. URL https://arxiv.org/abs/1907.10641v2.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv: 1911.02150*, 2019. URL https://arxiv.org/abs/1911.02150v1.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv: 2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202v1.

Shuai, X., Wang, Y., Wu, Y., Jiang, X., and Ren, X. Scaling law for language models training considering batch size. *arXiv preprint arXiv: 2412.01505*, 2024. URL https://arxiv.org/abs/2412.01505.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. Slimpajama: A 627b token cleaned and deduplicated version of redpajama, 2023.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv: 2104.09864*, 2021.

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.

Sun, Y., Dong, L., Zhu, Y., Huang, S., Wang, W., Ma, S., Zhang, Q., Wang, J., and Wei, F. You only cache once: Decoder-decoder architectures for language models. *Neural Information Processing Systems*, 2024. doi: 10.48550/arXiv.2405.05254. URL https://arxiv.org/abs/2405.05254v1.

Tian, K., Jiang, Y., Yuan, Z., Peng, B., and Wang, L. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *Neural Information Processing Systems*, 2024. doi: 10.48550/arXiv.2404.02905. URL https://arxiv.org/abs/2404.02905v2.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *NIPS*, 2017.

Waleffe, R., Byeon, W., Riach, D., Norick, B., Korthikanti, V., Dao, T., Gu, A., Hatamizadeh, A., Singh, S., Narayanan, D., Kulshreshtha, G., Singh, V., Casper, J., Kautz, J., Shoeybi, M., and Catanzaro, B. An empirical study of mamba-based language models. *arXiv preprint arXiv: 2406.07887*, 2024. URL https://arxiv.org/abs/2406.07887v1.

Wang, X. and Aitchison, L. How to set adamw's weight decay as you scale model and dataset size. *arXiv preprint arXiv: 2405.13698*, 2024. URL https://arxiv.org/abs/2405.13698.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv: 2406.01574*, 2024. URL https://arxiv.org/abs/2406.01574v4.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Xia, F., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2201.11903v6.

Wen, K., Dang, X., and Lyu, K. Rnns are not transformers (yet): The key bottleneck on in-context retrieval. *arXiv preprint arXiv: 2402.18510*, 2024. URL https://arxiv.org/abs/2402.18510v1.

Wortsman, M., Liu, P. J., Xiao, L., Everett, K. E., Alemi, A. A., Adlam, B., Co-Reyes, J. D., Gur, I., Kumar, A., Novak, R., Pennington, J., Sohl-Dickstein, J., Xu, K., Lee, J., Gilmer, J., and Kornblith, S. Small-scale proxies for large-scale transformer training instabilities. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ.

Wu, H. and Tu, K. Layer-condensed kv cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11175–11188, 2024.

Wu, J., Wang, Z., Zhang, L., Lai, Y., He, Y., and Zhou, D. Scope: Optimizing key-value cache compression in long-context generation. *arXiv preprint arXiv:2412.13649*, 2024.

Xu, H., Peng, B., Awadalla, H., Chen, D., Chen, Y.-C., Gao, M., Kim, Y. J., Li, Y., Ren, L., Shen, Y., Wang, S., Xu, W., Gao, J., and Chen, W. Phi-4-mini-reasoning: Exploring the limits of small reasoning language models in math. *arXiv preprint arXiv: 2504.21233*, 2025. URL https://arxiv.org/abs/2504.21233.

Xu, M., Men, X., Wang, B., Zhang, Q., Lin, H., Han, X., and Chen, W. Base of rope bounds context length. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Yan, Y., Chen, J., Qi, W., Bhendawade, N., Gong, Y., Duan, N., and Zhang, R. El-attention: Memory efficient lossless attention for generation. In *International Conference on Machine Learning*, pp. 11648–11658. PMLR, 2021.

Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., and Gao, J. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv: 2203.03466*, 2022. URL https://arxiv.org/abs/2203.03466v2.

Yang, G., Yu, D., Zhu, C., and Hayou, S. Tensor programs vi: Feature learning in infinite-depth neural networks. *International Conference on Learning Representations*, 2023. doi: 10.48550/arXiv.2310.02244. URL https://arxiv.org/abs/2310.02244v5.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear transformers with the delta rule over sequence length. *Neural Information Processing Systems*, 2024. doi: 10.48550/arXiv.2406.06484. URL https://arxiv.org/abs/2406.06484v1.

Yang, S., Kautz, J., and Hatamizadeh, A. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Ye, T., Dong, L., Xia, Y., Sun, Y., Zhu, Y., Huang, G., and Wei, F. Differential transformer. *arXiv preprint arXiv:2410.05258*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *Annual Meeting of the Association for Computational Linguistics*, 2019. doi: 10.18653/v1/P19-1472. URL https://arxiv.org/abs/1905.07830v1.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Neural Information Processing Systems*, 2019. doi: 10.5167/UZH-177483. URL https://arxiv.org/abs/1910.07467v1.

Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I. N. B., Zhan, H., He, J., Paul, I., Brunner, S., Gong, C., Hoang, J., Zebaze, A. R., Hong, X., Li, W., Kaddour, J., Xu, M., Zhang, Z., Yadav, P., and et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

A. Additional Aspect Ratio Calculations

Based on the Samba+YOCO architecture, we can derive the iso-parametric equation through calculating the number of non-embedding parameters as follows.

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/2,$$

$$N_{\text{mamba}}(d) = 6dw^2/4,$$

$$\begin{aligned} N(d) &= N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) \\ &= 208\alpha d^3 + 13.5\alpha^2 d^3 = 237568d^3. \end{aligned}$$

Solving for α, we get $\alpha_2 \approx 126$. For the SambaY+DA architecture, the aspect ratio is not changed because the number of extra parameters introduced by DA is negligible. For MambaY, we have

$$N_{\text{attn}}(d) = 2dw \cdot w_{\text{attn}}/4, \quad N_{\text{mamba}}(d) = 6dw^2/2,$$

$$N_{\text{gmu}}(d) = 4dw^2/4,$$

$$\begin{aligned} N(d) &= N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) \\ &= 64\alpha d^3 + 16\alpha^2 d^3 = 237568d^3. \end{aligned}$$

Solving for α, we get $\alpha_3 \approx 120$. For SambaY-MLP, we have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/4,$$

$$N_{\text{mamba}}(d) = 6dw^2/4, \quad N_{\text{gmu}}(d) = 8dw^2/4,$$

$$\begin{aligned} N(d) &= N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) \\ &= 144\alpha d^3 + 15.5\alpha^2 d^3 = 237568d^3. \end{aligned}$$

Solving for α, we get $\alpha_4 \approx 120$. For SambaY-Attn, we have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/4,$$

$$N_{\text{mamba}}(d) = 6dw^2/4, \quad N_{\text{gmu}}(d) = 2dw \cdot w_{\text{attn}}/4,$$

$$\begin{aligned} N(d) &= N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) \\ &= 208\alpha d^3 + 13.5\alpha^2 d^3 = 237568d^3. \end{aligned}$$

Solving for α, we get $\alpha_5 \approx 126$, which is the same as Samba+YOCO. For SambaY-Attn-All, we similarly have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4, \quad N_{\text{mamba}}(d) = 6dw^2/4,$$

$$N_{\text{gmu}}(d) = 2dw \cdot w_{\text{attn}}/2,$$

$$\begin{aligned} N(d) &= N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) \\ &= 208\alpha d^3 + 13.5\alpha^2 d^3 = 237568d^3. \end{aligned}$$

Solving for α, we get $\alpha_6 \approx 126$.

B. Ablation Study on Hyper-parameter Scaling Laws

We conduct a comprehensive ablation study of our μP++ scaling laws to validate their scaling behavior. All experiments are performed using Transformer++ trained with a 4K sequence length on the SlimPajama dataset. To ensure that the linear learning rate scheduler fully decays to zero, we train six models at different training token budgets: $\{100\text{B}, 200\text{B}, \ldots, 600\text{B}\}$ for each of the scaling curves. We examine the scaling performance under both tied and untied embedding setups. For the untied setting, we follow RWKV (Peng et al., 2023) by applying normal initialization with zero mean and standard deviation of 10^{-4}. The unembedding layer is initialized to zero, following the zero-out trick proposed in μP (Yang et al., 2022). We first explore batch size scaling with respect to training token size, following (Shuai et al., 2024; Li et al., 2025), *i.e.*

$$B = B_0\sqrt{\frac{T}{T_0}}.$$

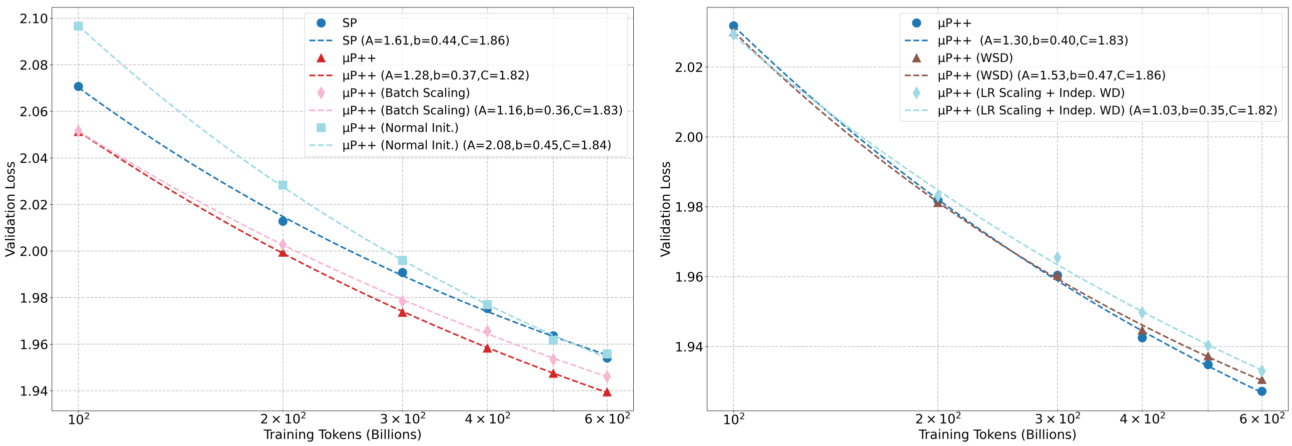
As in Figure 5a, μP++ (Batch Scaling) shows both worse learning efficiency and irreducible loss than μP++. Generally, we think the batch size mainly affects parallelism and the computation efficiency as long as the batch size is not too large, and the worse scaling behavior can be because (1) when scaling up, the batch size can surpass the critical batch size (McCandlish et al., 2018), which leads to worse model performance, (2) other optimizer hyper-parameters are not adjusted accordingly with batch size as in (Malladi et al., 2022) and we leave it for future works to study the large batch size training with μP++. We also try using Normal Initialization with 0.02 standard deviation for the weight matrices, and scale the variance with respect to $1/d$. However, μP++ (Normal Init.) shows worse scaling than μP++, indicating that it is better to adjust the initialization scaling based on each matrix' dimension as adopted by LeCun initialization, rather than a global factor related to model width. We explore integrating the empirical scaling law of the learning rate η scaling with respect to training tokens T (Bjorck et al., 2025) to μP++, *i.e.*,

$$\eta = \eta_0\sqrt{\frac{Bd_0}{B_0 d}}\left(\frac{T_0}{T}\right)^{\frac{1}{3}},$$

and adjust weight decay to maintain the same regularization effect across different training tokens with the setup of Independent Weight Decay (Wortsman et al., 2024), *i.e.*,

$$\lambda = \lambda_0\frac{\eta_0}{\eta},$$

where λ is the weight decay in AdamW (Loshchilov & Hutter, 2018) and $\lambda_0 = 0.1$. We denote this scaling law as



(a) Tied Embedding

(b) Untied Embedding

Figure 5: Validation Loss v.s. Training Tokens on the SlimPajama dataset for Transformer++ trained with tied (left) or untied (right) embedding layers.

μP++ (LR scaling + Indep. WD). As in Figure 5b, while the irreducible loss is comparable, we observe a worse learning efficiency with smaller b compared to μP++. We think that future work is needed to have an empirical study of the learning rate scaling with respect to dataset size under μP++, instead of transferring the empirical law directly to our theoretical laws. We also explore using the WSD (Hu et al., 2024) learning rate scheduler for μP++, where we set the final decay period to be $2/7$ of the total period following (DeepSeek-AI, 2024b). Unfortunately, it depicts worse scaling behavior than μP++ with a linear learning rate schedule, as shown in Figure 5b.

C. Additional Details on Scaling Comparisons

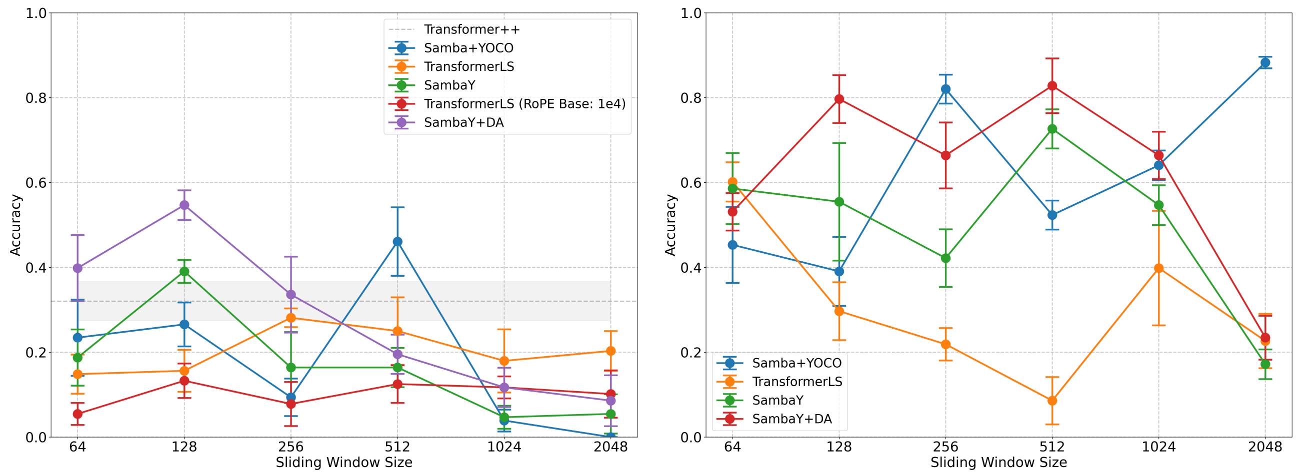
All models are trained with 4K sequence length for drawing the scaling curves. For Standard Parameterization (SP), we don't apply any μP++ scaling laws, and since LeCun initialization already scales its initialization variance with respect to $1/d_{in}$ as proposed in μP, where d_{in} is the fan-in dimension of the weight matrix, we use normal initialization with a standard deviation of 0.02 for weight matrices to rule out this confounding effect. We divide the initialization standard deviation of the output projection of each layer by $\sqrt{2d}$, following (Radford et al., 2019; Gu & Dao, 2023; Ren et al., 2025). The detailed architecture and optimization setups for each of the scales are shown in Table 6. Following (Gu & Dao, 2023; Yang et al., 2024; Ren et al., 2025; Yang et al., 2025), our downstream evaluations are conducted on the following benchmarks: Wikitext (Merity et al., 2016), LAMBADA (LMB) (Paperno et al., 2016), Arc-Easy/Challenge (ARC-e/ARC-c) (Clark et al., 2018), HellaSwag (Hella.) (Zellers et al., 2019), WinoGrande (Wino.) (Sakaguchi et al., 2021) and PIQA (Bisk et al., 2020), where we measure character normalized accuracy (acc_n) for Arc-Challenge and HellaSwag.

D. Additional Long-context Retrieval Experiments

Figure 6 illustrates how different model architectures perform on the Phonebook long-context task as the sliding window size increases, using either SlimPajama or ProLong-64K for pre-training with 32K sequence length and without variable-length training. Specifically, we concatenate the data samples with EOS tokens as separation to form 32K length training sequences. On SlimPajama, overall accuracy is modest, with SambaY+DA showing some initial promise at smaller window sizes (peaking at 128) before declining, while Samba+YOCO performs best at a moderate window size of 512. Transformer-based models generally struggle to achieve competitive accuracy across window sizes. Notably, reducing RoPE base from 640K to 10k for TransformerLS significantly harms the performance across window sizes. Switching to the ProLong-64K dataset leads to a notable performance boost across all architectures compared to SlimPajama, even without variable-length training. Notably, SambaY+DA achieves competitive accuracy using a smaller sliding window (512), matching the performance of Samba+YOCO at larger window sizes. While Samba+YOCO continues to benefit from increasing window sizes, reaching peak accuracy at 2048, SambaY+DA demonstrates greater efficiency by achieving strong results with smaller sliding window size. Given that variable-length training on ProLong-64K generally yields even better results as in Figure 3, these fixed-length training results serve as an important ablation. They highlight that while ProLong-64K benefits long-context performance,

Table 6: Architecture details for the model configurations explored in this work. TransformerLS adopts the same architecture as Transformer++, with Sliding Window Attention (SWA) applied to all attention layers except every fourth layer, which uses full attention. MLP Size denotes the intermediate dimension of the MLP, *i.e.*, the input dimension of the output projection. Phi4-mini and Phi4-mini-Flash are trained with a batch size of 8M tokens, using a linear learning rate schedule with 3,000 warm-up steps. The product of the head dimension and the number of query heads is not necessarily equal to the model width. Variants enhanced with Differential Attention adopt the same architectural configurations as their respective baselines. All models use tied embeddings. The 3.8B-parameter SambaY and Samba+YOCO models are randomly initialized for benchmarking under the vLLM inference framework.

Architecture	**Depth** d	**Model Width**	**Query Heads**	**KV Heads**	**Head Dim**	**MLP Size**	**Non-Embed Params (M)**	**Params (M)**	**Learning Rate**	**Training Tokens (B)**
Transformer++	8	1024	8	2	128	4096	121.6	154.4	5.66e-04	12.5
	12	1536	12	3	128	6144	410.5	459.7	4.62e-04	42.2
	16	2048	16	4	128	8192	973.1	1038.6	4.00e-04	100.0
	20	2560	20	5	128	10240	1900.5	1982.5	3.58e-04	195.3
	24	3072	24	6	128	12288	3284.1	3382.4	3.27e-04	337.5
SambaY	8	992	8	2	128	3968	123.3	155.0	5.66e-04	12.7
	12	1488	12	3	128	5952	416.1	463.7	4.62e-04	42.8
	16	1984	16	4	128	7936	986.3	1049.8	4.00e-04	101.4
	20	2480	20	5	128	9920	1926.5	2005.8	3.58e-04	198.0
	24	2976	24	6	128	11904	3328.9	3424.2	3.27e-04	342.1
Samba+YOCO	8	1008	8	2	128	4032	123.2	155.4	5.66e-04	12.7
	12	1512	12	3	128	6048	415.6	464.0	4.62e-04	42.7
	16	2016	16	4	128	8064	985.2	1049.7	4.00e-04	101.2
	20	2520	20	5	128	10080	1924.3	2004.9	3.58e-04	197.8
	24	3024	24	6	128	12096	3325.1	3421.9	3.27e-04	341.7
MambaY	16	1920	16	4	128	7680	975.2	1036.6	4.00e-04	40.0
MambaY-2	16	1920	16	4	128	7680	975.2	1036.6	4.00e-04	40.0
SambaY-2	16	1984	16	4	128	7936	986.3	1049.8	4.00e-04	40.0
SambaY-A	16	2016	16	4	128	8064	985.2	1049.7	4.00e-04	40.0
SambaY-AA	16	2016	16	4	128	8064	985.2	1049.7	4.00e-04	40.0
SambaY-MLP	16	1920	16	4	128	7680	985.0	1046.4	4.00e-04	40.0
Phi4-mini	32	3072	24	8	128	8192	3221.2	3835.8	5.00e-04	5000
Pih4-mini-Flash	32	2560	40	20	64	10240	3329.2	3841.4	5.00e-04	5000
SambaY	32	2560	40	20	64	10240	3329.2	3841.4	-	-
Samba+YOCO	32	2560	40	20	64	10240	3224.4	3736.5	-	-



(a) SlimPajama (b) ProLong-64K

Figure 6: Accuracy (with error bars) v.s. Sliding Window Size on Phonebook with 32K evaluation length using 40B training tokens from SlimPajama (left) or ProLong-64K (right). As an ablation to Figure 3, variable-length training is not applied for both setting.

the full potential, especially for models sensitive to sliding window size (e.g. TransformerLS), can be further unlocked by training methodologies that explicitly account for varying sequence lengths of each data sample. The varying optimal sliding window sizes and performance trajectories underscore that both the pre-training dataset and the training methodology significantly influence how effectively the training context length can be utilized for long-context pre-training.

E. More Details on Architecture and Large-scale Pre-training

We provide a comprehensive summarization of our architectures and large-scale pre-training setting in Table 6. In our architectures, Differential Attention uses a depth-dependent initialization factor, $\lambda_{\text{init}} = 0.8 - 0.6\exp(-0.3 \times l)$, where l is the depth index. For each attention head, it employs two sets of learnable parameters, $(\lambda_{q1}, \lambda_{k1})$ and $(\lambda_{q2}, \lambda_{k2})$, each of dimension equal to the head dimension and initialized with a normal distribution of zero mean and 0.1 standard deviation. RMSNorm (Zhang & Sennrich, 2019) with learnable element-wise affine parameters is adopted for attention output normalization.

F. Additional Details on Reasoning Results

Following Phi4-mini-Reasoning (Xu et al., 2025), the evaluation is conducted with a temperature setting of 0.6, a top-p value of 0.95, and a maximum sequence length of 32,768 tokens. Each task involves 3 runs, and the average performance across these trials is reported.

We conduct case studies for Phi4-mini-Flash-Reasoning in Example 1 and Example 2 for both mathematical reasoning and daily-life reasoning. Notably, while our model is only trained on distillation data with math problems, we can see our model exhibits advanced reasoning techniques such as double checking (*e.g. "Alternatively, maybe I can complete the square instead.", "Let's break it down:"*), reflection (*e.g. "I should also check if there was a mistake in the problem statement.", "Wait, maybe I'm miscounting."*), and backtracking (*e.g. "So factoring doesn't seem straightforward here. Maybe I need to use the quadratic formula instead.", "So between B and Y, there are three 'r's."*) in both cases. Surprisingly, in Example 2, our model will even guess the user intention (*e.g. "But wait, maybe the user intended "strawberry" which has two 'r's. But the question specifically says "strawberrry", so I have to go with that."*) to finally provide the correct answer.

Example 1: Mathematical Reasoning (Part 1)

Query: solve x^2+3x+4=1

Model's reasoning process:

Think

Okay, so I need to solve the equation x squared plus three x plus four equals one. Let me write that down first to make sure I have it right: $x^2 + 3x + 4 = 1$. Hmm, alright. The first thing I remember is that to solve a quadratic equation, it's usually best to get everything on one side so that the other side is zero. That way, I can factor it or use the quadratic formula if needed. Let me try subtracting 1 from both sides to move that 1 over. So, subtracting 1 from both sides gives me $x^2 + 3x + 4 - 1 = 0$. Simplifying that, 4 minus 1 is 3, so the equation becomes $x^2 + 3x + 3 = 0$.

Now, I need to check if this quadratic can be factored easily. Let me see. The standard form is $ax^2 + bx + c$, so here a is 1, b is 3, and c is 3. To factor it, I need two numbers that multiply to 3 (since a is 1, it's just c) and add up to 3. Let me think... The factors of 3 are 1 and 3. But 1 + 3 is 4, which is more than 3. There's also negative numbers, but since both the product and the sum are positive, both numbers need to be positive. But 1 and 3 add to 4, which isn't 3. So factoring doesn't seem straightforward here. Maybe I need to use the quadratic formula instead.

The quadratic formula is x equals negative b plus or minus the square root of b squared minus four a c all over 2a. Let me plug in the values. Here, a is 1, b is 3, and c is 3. So substituting those in, we get $x = [-3 \pm \sqrt{}(3^2 - 4*1*3)] / (2*1)$. Let me compute the discriminant first, which is the part under the square root: 3 squared is 9, and 4 times 1 times 3 is 12. So 9 minus 12 is -3. Wait, the discriminant is negative. That means there are no real solutions, only complex ones.

But the problem didn't specify whether they want real solutions or if complex solutions are acceptable. Since it's a math problem, unless specified, sometimes they expect complex solutions. Let me proceed. So the square root of -3 can be written as $i\sqrt{3}$, where i is the imaginary unit. Therefore, the solutions would be $x = [-3 \pm i\sqrt{3}]/2$.

Let me double-check my steps to make sure I didn't make a mistake. Starting with the original equation: $x^2 + 3x + 4 = 1$. Subtract 1: $x^2 + 3x + 3 = 0$. Check. Then discriminant: $b^2 - 4ac = 9 - 12 = -3$. Correct. So the roots are complex: $(-3 \pm i\sqrt{3})/2$.

Alternatively, maybe I can complete the square instead. Let me try that method to verify. Starting with $x^2 + 3x + 3 = 0$. Move the constant term to the other side: $x^2 + 3x = -3$. To complete the square, take half of the coefficient of x, which is 3/2, square it to get (9/4). Add that to both sides: $x^2 + 3x + 9/4 = -3 + 9/4$. The left side becomes $(x + 3/2)^2$. The right side: -3 is -12/4, so -12/4 + 9/4 is -3/4. So $(x + 3/2)^2 = -3/4$. Taking the square root of both sides gives $x + 3/2 = \pm\sqrt{}(-3/4) = \pm i\sqrt{}(3)/2$. Subtract 3/2 from both sides: $x = -3/2 \pm i\sqrt{3}/2$. Which is the same as $(-3 \pm i\sqrt{}3)/2$. So that matches the quadratic formula result.

Therefore, the solutions are complex numbers: $(-3 + i\sqrt{3})/2$ and $(-3 - i\sqrt{3})/2$. Since the original equation is a quadratic, there are two solutions, and they are complex conjugates of each other.

I should also check if there was a mistake in the problem statement. The original equation was $x^2 + 3x + 4 = 1$. If I plug the solutions back into the equation, they should satisfy it. Let me test one of them. Let's take $x = [-3 + i\sqrt{3}]/2$. Plugging into $x^2 + 3x + 4$. First, compute x^2:

$x = (-3 + i\sqrt{3})/2$. So x squared is $[(-3 + i\sqrt{3})/2]^2$. Let's compute that. First, square the numerator: $(-3 + i\sqrt{3})^2$. That's $(-3)^2 + 2*(-3)*(i\sqrt{3}) + (i\sqrt{3})^2$. Which is $9 - 6i\sqrt{3} + (i^2*(\sqrt{3})^2)$. Since i^2 is -1, that term becomes -1*3 = -3. So $9 - 6i\sqrt{3} -3 = 6 - 6i\sqrt{3}$. Then divide by 2^2=4. So $x^2 = (6 - 6i\sqrt{3})/4 = (3 - 3i\sqrt{3})/2$.

Now compute 3x: $3*[(-3 + i\sqrt{3})/2] = (-9 + 3i\sqrt{3})/2$.

Adding $x^2 + 3x + 4$: $(3 - 3i\sqrt{3})/2 + (-9 + 3i\sqrt{3})/2 + 4$. Combine the fractions: $[3 - 3i\sqrt{3} -9 + 3i\sqrt{3}]/2 + 4$. The imaginary parts cancel out: (3 -9)/2 +4 = (-6)/2 +4 = -3 +4 = 1. Which matches the right-hand side of the equation. So that works. Similarly, the other root would also satisfy the equation.

Therefore, the solutions are correct. So the answer is x equals (-3 plus or minus $i\sqrt{3}$) over 2.

I think that's all. The key steps were moving 1 to the left to set the equation to zero, recognizing that the quadratic doesn't factor nicely, computing the discriminant, finding it's negative, and then proceeding with the quadratic formula or completing the square to find the complex solutions. Then verifying by plugging back in. So the final answer should be the two complex numbers.

Model's Response:

To be continued...

Example 1: Mathematical Reasoning (Part 2)

Model's Response (continued):
To solve the equation $x^2 + 3x + 4 = 1$:
1. **Rearrange the equation** to standard quadratic form:

$$x^2 + 3x + 4 - 1 = 0 \implies x^2 + 3x + 3 = 0$$

2. **Identify coefficients**: $a = 1$, $b = 3$, $c = 3$.
3. **Compute the discriminant**:

$$\Delta = b^2 - 4ac = 3^2 - 4(1)(3) = 9 - 12 = -3$$

Since the discriminant is negative, the solutions are complex.
4. **Apply the quadratic formula**:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} = \frac{-3 \pm \sqrt{-3}}{2} = \frac{-3 \pm i\sqrt{3}}{2}$$

5. **Verify by completing the square**:

$$x^2 + 3x = -3 \implies \left(x + \frac{3}{2}\right)^2 = -\frac{3}{4}$$

Taking square roots:

$$x + \frac{3}{2} = \pm\frac{i\sqrt{3}}{2} \implies x = -\frac{3}{2} \pm \frac{i\sqrt{3}}{2}$$

This confirms the solutions.
Solutions:

$$x = \frac{-3 + i\sqrt{3}}{2} \quad \text{and} \quad x = \frac{-3 - i\sqrt{3}}{2}$$

$$\boxed{x = \frac{-3 \pm i\sqrt{3}}{2}}$$

Example 2: Letter Counting in Misspelled Words

Query: how many r's in "strawberrry"?

Model's reasoning process:

Think

Okay, let's see. The question is asking how many 'r's are in the word "strawberrry". Alright, first I need to look at the word carefully. Let me write it out: S-T-R-A-W-B-E-R-R-Y. Wait, the original word is "strawberrry". So, the spelling has three 'r's in a row, right? Let me count them one by one.
Starting from the beginning: S (1), T (2), R (3), A (4), W (5), B (6), E (7), R (8), R (9), Y (10). Hmm, so between the B and Y, there are three 'r's. Let me check again. The word is spelled as S-T-R-A-W-B-E-R-R-R-Y. So yes, after the B and E, there are three 'r's. So that's three 'r's in a row. Wait, but when I count the letters, let me make sure I'm not missing any. Let's break it down:
S T R A W B E R R R Y Wait, so actually, there are four 'r's? Wait, no. Wait, the original word is "strawberrry". Let me count the letters in "strawberrry". Let's spell it out: S-T-R-A-W-B-E-R-R-R-Y. Let's count each letter:
S T R A W B E R R R Y So that's 11 letters total. Now, the 'r's are at positions 3, 8, 9, 10. So that's four 'r's? Wait, but the word is written as "strawberrry". So maybe the user intended three 'r's? Wait, but when I look at the spelling, "strawberry" normally has two 'r's. But here it's "strawberrry" with three 'r's. Wait, maybe the correct spelling is "strawberry" with two 'r's. But the user wrote "strawberrry" with three 'r's. So in that case, the answer would be three 'r's. Wait, but when I count the letters in "strawberrry", let's see: S-T-R-A-W-B-E-R-R-R-Y. So that's three 'r's in a row. So total 'r's would be three. Wait, but when I count the letters, positions 3, 8, 9, 10. Wait, that's four 'r's. Wait, maybe I'm miscounting. Let me write the word out:
S (1), T (2), R (3), A (4), W (5), B (6), E (7), R (8), R (9), R (10), Y (11). So positions 3,8,9,10 are 'r's. That's four 'r's. Wait, but the word is "strawberrry". Let me check the spelling again. The user wrote "strawberrry". So between B and Y, there are three 'r's. So the word is spelled as S-T-R-A-W-B-E-R-R-R-Y. So that's three 'r's after the B and E. So total 'r's: the first 'r' is at position 3, then three more at positions 8,9,10. Wait, that's four 'r's in total. Wait, but maybe the user made a typo. Let me confirm. The word is "strawberrry". So "straw" + "berrry". So "straw" is S-T-R-A-W, then "berrry" would be B-E-R-R-R-Y. So in "berrry", there are three 'r's. So total in the whole word: the first 'r' in "straw", then three 'r's in "berrry". So total of four 'r's. Wait, but the original word is "strawberrry". Let me check again. S-T-R-A-W-B-E-R-R-R-Y. So yes, four 'r's. But maybe the user intended "strawberry" with two 'r's, but wrote "strawberrry" by mistake. But the question is as given, so I should answer based on the exact spelling provided. So in "strawberrry", how many 'r's? Let's count each occurrence:
Looking at each letter:
s, t, r, a, w, b, e, r, r, r, y.
So the 'r's are the third letter, then the eighth, ninth, and tenth letters. That's four 'r's. Wait, but maybe I'm miscounting. Let me list them:
S T R A W B E R R R Y Yes, so positions 3,8,9,10. That's four 'r's. So the answer should be 4. But wait, maybe the user intended "strawberry" which has two 'r's. But the question specifically says "strawberrry", so I have to go with that. Therefore, the number of 'r's is 4. Wait, but let me check once more. Let's spell it out: S-T-R-A-W-B-E-R-R-R-Y. So after B and E, there are three 'r's. So that's three 'r's in the middle, plus the initial 'r' in "straw". So total of four. Yes. So the answer is 4.

Model's Response:
The word "strawberrry" is spelled as S-T-R-A-W-B-E-R-R-R-Y. Breaking it down:
Letters: S, T, R, A, W, B, E, R, R, R, Y 'r's: The third letter is 'r', followed by three more 'r's later in the word. Total number of 'r's: 4.
$\boxed{4}$

G. Related Works

KV Cache Sharing. Efficient inference in transformer-based models has been significantly advanced through techniques that reduce memory consumption, particularly concerning key-value (KV) caching. Traditional approaches like Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) have enabled multiple query heads to share a single key/value head within the same layer, effectively reducing the number of distinct key/value heads with minimal impact on accuracy. Apart from YOCO (Sun et al., 2024), Cross-Layer Attention (CLA) (Brandon et al., 2024) extends KV sharing across adjacent layers, achieving up to two times reduction in KV cache size while maintaining performance. Our work focuses on representation sharing across SSM layers through directly sharing the output from the SSM kernel to avoid materializing recurrent states, thereby preserving the parallel training efficiency of SSM layers.

Efficient Long Generation. Efficient long-sequence generation in transformer models has been a focus of recent research on LLM efficiency, primarily due to the substantial memory demands associated with key-value (KV) caching during inference with long CoTs (Kitaev et al., 2020; Wu et al., 2024; Yan et al., 2021; Duanmu et al., 2024). To address these challenges, several techniques have been proposed to optimize memory usage without compromising model performance. One notable approach is the Layer-Condensed KV Cache (LCKV) (Wu & Tu, 2024), which computes and caches KV pairs for only a subset of layers, significantly reducing memory consumption and improving inference throughput. Another advancement is InfiniGen (Lee et al., 2024), a dynamic KV cache management framework that selectively prefetches essential KV cache entries, thereby mitigating fetch overhead from host memory in offloading-based LLM serving systems. These methods collectively contribute to more efficient long-sequence generation by optimizing KV cache usage, and are orthogonal to our works, as we can also apply these techniques to improve the memory I/O efficiency of our full attention layer.

Neural Scaling Laws. Understanding how model performance scales with size and data is crucial for efficient large-scale training. Empirical studies have shown that transformer models exhibit predictable scaling behaviors, where performance improves with increased model parameters and training data (Hestness et al., 2017; Kaplan et al., 2020; Bahri et al., 2024; Alabdulmohsin et al., 2022; Hoffmann et al., 2022). Numerous works have also investigated scaling laws for hyper-parameters, based on either empirical studies (Bjorck et al., 2025; Wortsman et al., 2024) or theoretical analyses (Malladi et al., 2022; Yang et al., 2022; 2023; Wang & Aitchison, 2024). In this work, we focus on theoretical hyper-parameter scaling laws since they are not over-tuned for the Transformer architectures, so they could provide fairer comparisons for the emerging architectures.

H. Limitation

We validate our model's reasoning capability using distillation-based Supervised Fine-Tuning (SFT), but Reinforcement Learning (RL) remains under-explored in the context of hybrid architectures. Due to resource constraints, we do not perform an exhaustive hyperparameter search for each architecture. Instead, we adopt a generic optimization setup based on Transformer++ for learning rate, weight decay, warm-up schedule, batch size, AdamW betas, epsilon, and other parameters. It is likely that aggressive tuning of these optimization settings could yield improved results. We leave a more comprehensive study of the interplay between optimization setups and architecture designs for future work. Lastly, our architecture still includes a full-attention layer, which leads to linear computational complexity during decoding. This underscores future research direction on designing models for extremely long sequence generation that can maintain constant decoding complexity while effectively leverage long-context memory.