# SHISHULM: Achieving Optimal and Efficient Parameterization with Low Attention Transformer Models

**Anonymous ACL submission**

## Abstract

While the transformer architecture has achieved state-of-the-art performance on natural language processing tasks, these models impose substantial memory and computational overhead. Recent research has identified significant architectural redundancies within these models, particularly in the attention sub-layers in the top layers, presenting opportunities for optimization without compromising performance. Taking insights from research on inference-time layer pruning and depth-dependent computation in language models, we introduce an efficient language model architecture referred to as SHISHULM. By replacing full decoder layers at the top of the model with MLP-only blocks, we achieve up to $10-60\%$ improvement in generation latency and $1.3-5\times$ gain in throughput. Upon further sharing parameters across adjacent MLP-only layers of SHISHULM, we obtain up to 20% savings in memory with minimal degradation in performance. Our findings provide insights towards building more efficient language modeling architectures from a pre-training standpoint by leveraging how information flows in transformers.

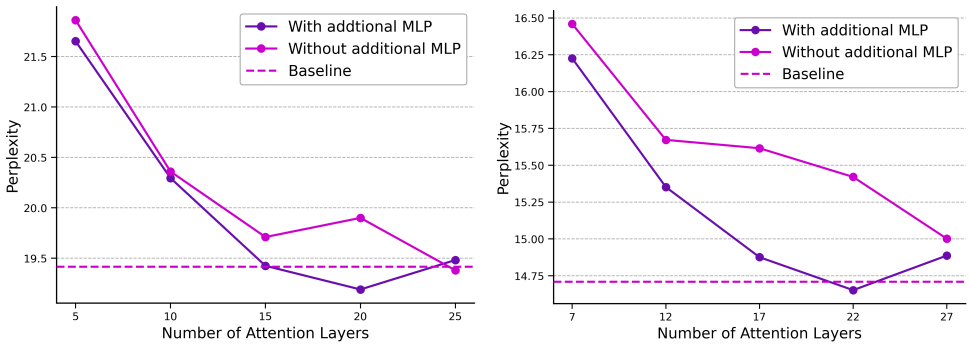
## 1 Introduction

The transformer model architecture introduced by Vaswani et al. (2017) has emerged as the dominant paradigm for language modeling, achieving state-of-the-art performance across a broad range of natural language processing tasks. Decoder-only transformers consist of a stack of identical layers (blocks), each comprising a multi-head self-attention (MHSA) sub-layer followed by a position-wise feed-forward (MLP) sub-layer, with residual connections and layer normalization applied throughout. In the self-attention mechanism, each token attends over all preceding tokens by computing queries, keys, and values from the hidden representation, enabling rich contextual interactions across the sequence. The MLP sub-layer then applies a non-linear transformation independently at each token position, modulating the resulting representations.

Despite their impressive capabilities, Large Language Models (LLMs) typically contain billions of trainable parameters (Bai et al., 2023; Touvron et al., 2023; Jiang et al., 2023; Javaheripi et al., 2023), imposing substantial memory and compute requirements. The sheer parameter count necessitates significant memory for storage alone. During training, this is compounded by the need to retain intermediate activations for backpropagation. During inference, the attention mechanism requires maintaining a key-value (KV) cache that grows linearly with both sequence length and the number of attention layers, further increasing memory pressure.

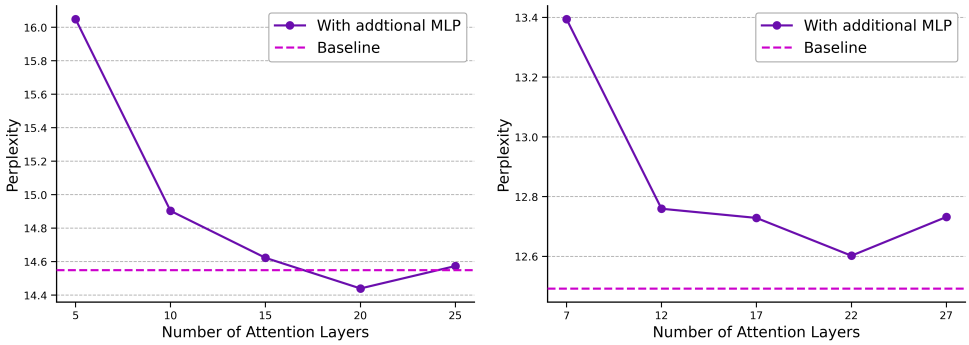
However, several lines of research have identified significant redundancies. At inference time, structured and unstructured pruning methods remove superfluous parameters from pre-trained models (Ma et al., 2023; Frantar and Alistarh, 2023), while layer-level pruning identifies and eliminates entire decoder layers using metrics such as cosine similarity (Men et al., 2024; Gromov et al., 2025), Shapley values (Siddiqui et al., 2024), or perplexity on a calibration set (Song et al., 2024). However, these approaches inherit the full pre-training costs of the original architecture and incur additional overhead for identifying components to prune and performance recovery through fine-tuning.

A separate line of research focuses on reducing the Key-Value (KV) cache requirements by pre-training models with the key-value pairs shared across different layers. For instance, Wu and Tu compute keys and values only at a later layer and share them across bottom layers, whereas Sun et al. compute key-value pairs only for the first half of the layers and share them across the remaining layers.




(a) 125M (b) 350M

Figure 1: Effect of removing attention layers and adding extra MLP layers to regain performance. Models in (a) and (b) are trained on 5B and 10B tokens, respectively.




(a) 125M (b) 350M

Figure 2: Effect of removing attention layers and adding extra MLP layers, trained on 50B tokens.

A growing body of evidence suggests that these costs are not fully justified by the computational necessity of each architectural component. For instance, (Men et al., 2024) and Gromov et al. (2025) both observe that later decoder blocks in LLaMA-style architectures exhibit high cosine similarity between their block inputs and outputs, suggesting near-identity transformations in those layers.

More directly relevant to our work, studies on the *no-op* behaviour of attention (Bondarenko et al., 2023; Barbero et al., 2025) show that, in many heads, the attention pattern collapses to attending almost exclusively to a single token (often the first), effectively making the entire operation a pass-through. This is consistent with the observation of *attention sinks* (Xiao et al., 2024), where certain semantically meaningless tokens absorb disproportionate attention.

Furthermore, research in AI interpretability has emphasized *depth-wise information flow* in transformers. Early layers primarily perform *token mixing*, integrating contextual information across the sequence, while later layers largely *refine the output probability distribution*, sharpening predictions (Lad et al., 2024; Csordás et al., 2025; de Llano et al., 2026).

Building on these insights, we ask the following question:

> Is there a more optimal allocation of parameters that is both high performing and efficient?

We answer this in the affirmative. Our experiments reveal that pre-training decoder-only transformers by replacing full decoder layers at the top with MLP-only sub-layers performs comparably to the full decoder baseline at the same parameter scale, while achieving reductions in training and inference costs. Furthermore, leveraging high weight distribution similarity between adjacent MLP-only layers, we share weights across these layers in pairs, achieving further memory savings with minimal performance loss. This work is, to the best of our knowledge, among the first to directly translate insights from interpretability research into an *architectural design choice at pre-training time*. We use the MobileLLM model family (Liu et al., 2024),

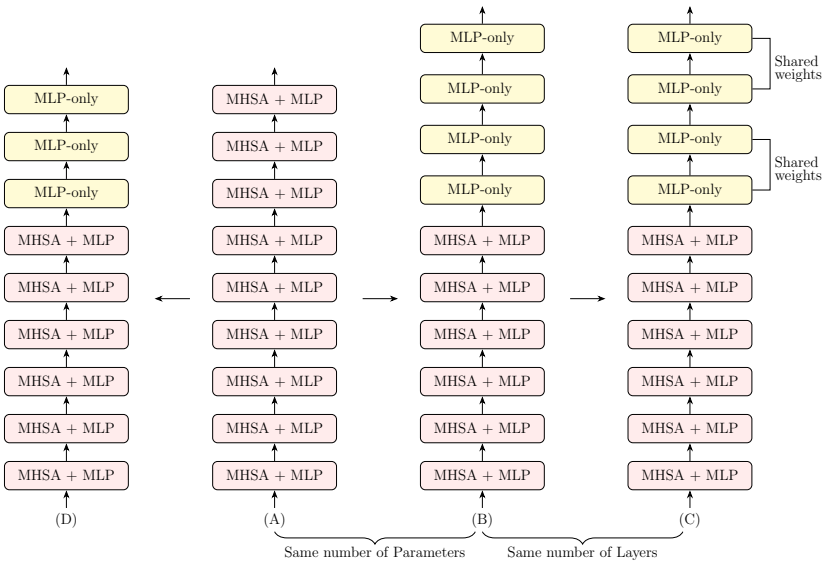



Figure 3: This figure describes the architectures on which we perform our pre-training experiments. (A) shows Vanilla Transformer model with each layer as a full-decoder layer (MHSA + MLP), (D) is the result of removing MHSA from the top layers. (B) is obtained by removing attention layers from (A) and then adding extra MLPs to balance out parameters. (C) is obtained from (B) by tying together the weights of the MLP-only layers at the top, in pairs of two.

with parameter counts ranging from 125M to 1.5B, to validate the proposed architecture.

**Key contributions.**

1. We propose SHISHULM[1], an efficient architecture for pre-training decoder-only transformer-based language models that employs only MLP sub-layers in the later blocks of the model, motivated by depth-wise redundancy in attention computation.

2. We validate SHISHULM by demonstrating comparable downstream performance and improved efficiency relative to the MobileLLM baselines at scales ranging from 125M to 1.5B parameters.

3. We further present a *weight-shared* variant of SHISHULM, in which parameters are shared across adjacent MLP-only layers, achieving substantial memory savings with negligible performance degradation.

## 2 Preliminaries

### 2.1 Baseline architecture

We use decoder-only transformer models as our baselines. These models consist of identical decoder layers that are stacked sequentially. Each of the decoder layers further consists of a Mutli-Headed-Self-Attention (MHSA) sub-layer followed by a Multi Layer Perceptron (MLP) sub-layer. The MHSA and MLP layers are connected in a residual fashion, i.e., the inputs are added to the outputs before being passed to the next layer. We study pre-LayerNorm architectures in this work, where the inputs are normalized before passing to each of the MHSA and MLP sub-layers. Let $L$ denote the number of decoder layers in the model, and $x_i$ be the input to the $i^{th}$ block. The forward computation is given by

$$x_{i+1} = y_i + \text{ MLP } ( \text{ LN } (y_i) \ )$$

where,

$$y_i = x_i + \text{MHSA } ( \text{ LN } (x_i) \ ).$$

MLP(.), LN(.), and Self-Attention(.) represent the functions computed by MLP, normalization and self-attention layers, respectively.

### 2.2 Depth-wise Information Flow in Transformers

A growing body of interpretability research has established that computation in transformer networks is not uniform across depth, but rather proceeds in functionally distinct *stages* (Lad et al., 2024; de Llano et al., 2026). Lad et al. identify that

[1]The word *Shishu* in Sanskrit means "baby."

inference in LLMs occurs in four distinct, depth-dependent stages: (1) detokenization, (2) feature engineering, (3) prediction ensembling, and (4) residual sharpening. Crucially, they observe that the models exhibit remarkable robustness to interventions such as layer skipping and swapping in intermediate layers. de Llano et al. propose a *Mix-Compress-Refine framework*: Early layers (0–20% depth) mix information broadly via diffuse attention. Middle layers (20–85%) compress representations while halting mixing through attention sinks and later layers (85–100%) selectively refine through localized attention

Csordás et al. provide complementary evidence by comparing representations across models of different sizes, finding that layers at similar *relative* depths map most closely to one another across architectures. This suggests that larger models do not learn qualitatively new types of computation, but rather distribute the same functional stages more finely across more layers. They further observe that later half of the model spends a disproportionate fraction of model capacity performing distribution matching rather than integrating new information and composition, characterizing this as a potentially wasteful use of parameters.

Artzy and Schwartz reveal a depth-dependent division of attention mechanisms, showing that early layers primarily use attention for gathering information from previous tokens, while the later layers shift towards internal consolidation with significantly reduced attention importance. Their findings indicate that manipulating representations in the top 30-50% of layers has minimal performance impact, supporting the hypothesis that layers serve fundamentally different computational purposes with changing depth.

Together, these findings motivate a simple design principle: attention is crucial in the early layers where information integration across tokens is the primary computational task, but might become redundant in later layers since they mostly perform refinement of the output distribution. We therefore attempt to explore if pre-training can be done without these sub-layers in the first place. SHISHULM is a direct embodiment of this principle.

## 3 Experiments

In this section, we describe our pre-training experiments.

**Setup.** Due to their strong performance at smaller scales, we use the MobileLLM model family (Liu et al., 2024), with parameter counts ranging from 125M to 1.5B, as our base architectures. These models follow the LLaMA architecture (Touvron et al., 2023). All pre-training experiments are conducted on 8 H200 GPUs using subsets of the SlimPajama dataset (Soboleva et al., 2023). Additional training details are provided in Appendix C.

### 3.1 Pre-training with Fewer Attention Layers

Motivated by the evidence of redundancy in attention computation in the later layers, we first investigate the impact of removing attention sub-layers from the top layers of 125M and 350M MobileLLM models, in steps of 5 layers. We train these modified models on 5B and 10B tokens, respectively, and measure validation perplexity. We find that naively removing attention sub-layers leads to a degradation in performance, as shown in Figure 1. To regain performance, we attempt a simple corrective strategy: adding MLP sub-layers (with the same dimensions as the original model) towards the top to approximately restore the original parameter count.

In essence, we still remove attention layers in steps of 5, but also add sufficient number of MLPs so that the parameters are approximately 125M and 350M (same as the parent models). We pre-train these models on the same data for each scale, and these parameter-matched models perform on par with the full-attention baselines as shown in Figure 1. However, when the fraction of retained attention layers drops below roughly $50\%$, we observe significant performance degradation.

To validate this finding at larger data scales and reduce the effects of stochastic noise in training, we extended training of these 125M and 350M models to 50B tokens, along with the parent models. The same pattern holds (as shown in Figure 2): retaining up to $2/3$ of the attention layers with sufficient compensating MLPs matches or slightly surpasses baseline perplexity and is the most optimal among models with fewer attention parameters. We adopt this $2/3$ attention retention configuration and apply it to the 600M, 1B, and 1.5B MobileLLM models, pre-training each on 50B tokens. Due to compute constraints we were unable to test other configurations on these scales. However, we observe that this configuration achieves performance similar to and in most cases, slightly better than the respective MobileLLM baselines, with the excep-

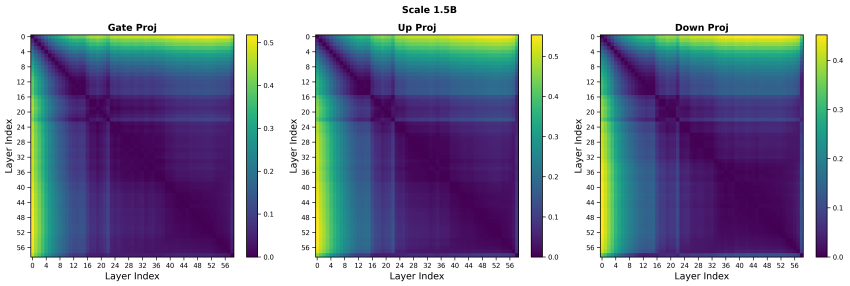



Figure 4: Earth-Mover's distance across all pairs of layers in SHISHULM-1.5B.

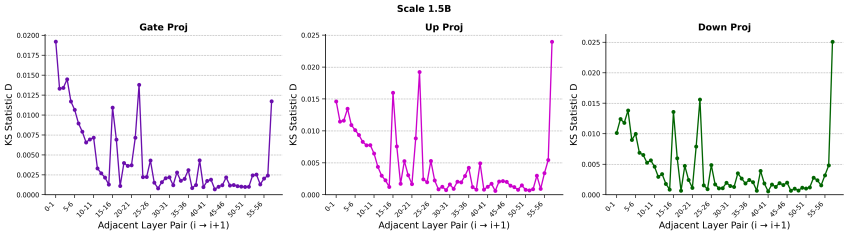



Figure 5: KS-similarity between adjacent layers in SHISHULM-1.5B.

| Parent model | Baseline | Shishu | Shishu-S |
|---|---|---|---|
| MobileLLM-125 | 14.55 | **14.44** | 14.67 |
| MobileLLM-350 | **12.49** | 12.60 | 12.91 |
| MobileLLM-600 | 11.70 | **11.65** | 12.00 |
| MobileLLM-1b | **11.03** | **11.03** | 11.30 |
| MobileLLM-1.5b | 10.67 | **10.54** | 10.75 |

Table 1: Validation perplexities after pre-training by applying our proposed changes to the base architectures. Lower is better.

tion of MobileLLM-350M (see Table 1). We call these models "Shishu Language Models" (abbv. SHISHULM). The architecture is shown in Figure 3. These findings reflect a more parameter-efficient allocation, with model capacity in later layers entirely directed towards MLP computation.

### 3.2 Weight Sharing Across MLP-only Layers

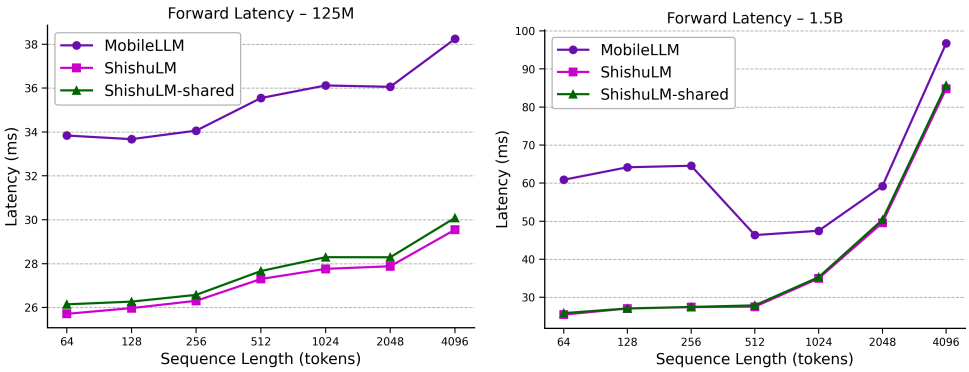
We hypothesize the loss of performance by naive removal of attention to one or both of the following:

1. **Loss of functional modules:** Removal of attention may eliminate computational pathways.

2. **Loss of parameters:** Simply removing layers reduces the total parameter count, diminishing model expressiveness.

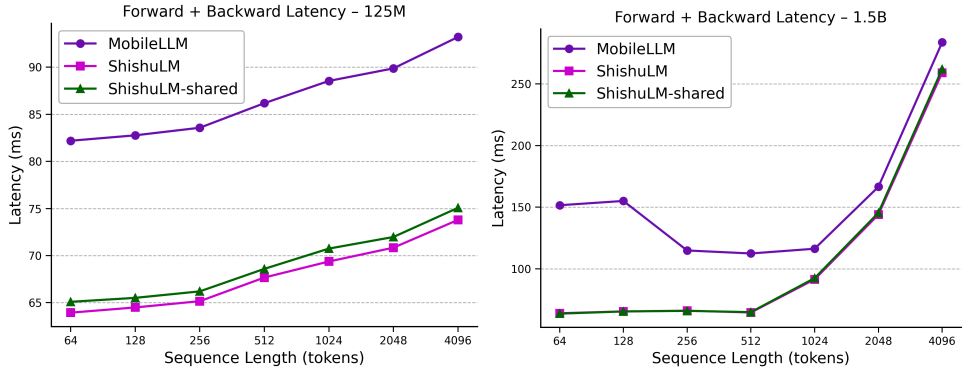
Although adding sufficient number of MLP sub-layers restores performance, it leads to both increment in the number of sub-layers and parameters, as compared to the models with no extra MLPs. It is therefore not clear as to which of the two factors contribute to the results observed in the previous sub-section. In this sub-section we attempt to reduce the parameter count of SHISHULM by sharing weights across the MLP-only layers. Our motivation is twofold. First, prior work supports shared weight and repeated computation (Csordás et al., 2025, 2024): larger models appear to "stretch" the same functional computations over more layers. Second, we conduct a weight distribution similarity analysis between adjacent MLP-only layers in our trained SHISHULM models. We measure the Earth Mover's Distance (EMD) across all pairs of layers and observe high similarity between the adjacent layers (See Figure 4). We validate these findings using the the Kolmogorov-Smirnov (KS) statistic, which captures the maximum point-wise deviation between two distributions. As shown in Figure 5, after an intermediate layer, where the dissimilarity peaks, adjacent layers have dissimilarity values very close to 0. Please refer to Appendix B for details on these metrics.

Moreover, since MLP sub-layers account for a significant fraction of total model parameters and contribute disproportionately to storage requirements, weight sharing across adjacent MLP-only layers in pairs of two offers parameter reduction while maintaining the similar depth. We pre-train weight-shared variants of all SHISHULM models (125M through 1.5B) by tying the parameters of every two consecutive MLP-only layers (including




(a) 125M (b) 1.5B

Figure 6: Forward pass latency (ms) of models at the (a) 125M and (b) 1.5B scale.




(a) 125M (b) 1.5B

Figure 7: Forward + backward pass latency (ms) of models at the (a) 125M and (b) 1.5B scale.

the normalization layer). When trained on 50B tokens, the degradation remains minimal (see Table 1). We refer to these models by SHISHULM-X-S, where X indicates the number of parameters in the parent MobileLLM model.

Since the SHISHULM-S models have fewer parameters than the original MobileLLM baselines, this provides additional evidence for the first hypothesis: the performance degradation from naive attention removal is largely attributable to the loss in computational blocks rather than parametric expressibility, and a few extra MLP layers can account for the computation done by the removed attention sub-layers.

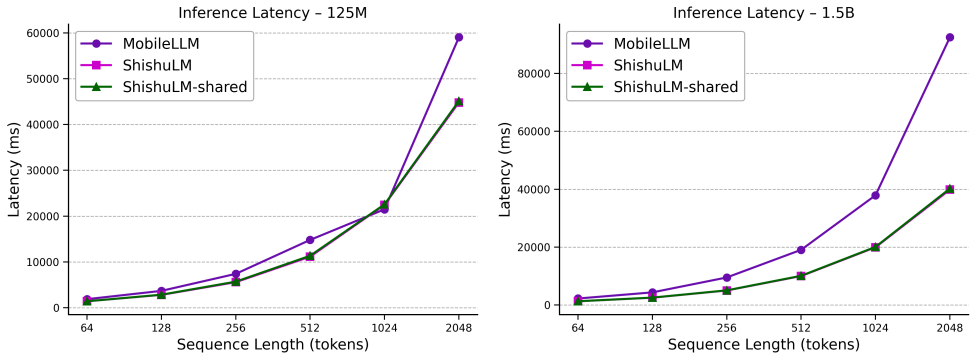
## 4 Downstream Performance

We evaluate all our pretrained models on a suite of standard zero-shot commonsense reasoning benchmarks using the EleutherAI LM Evaluation Harness (Gao et al., 2023): HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), WinoGrande (Sakaguchi et al., 2021), ARC-Easy and ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), Social-IQA (Sap et al., 2019) and PIQA (Bisk et al., 2020) in Table 3. We also report perplexity on the WikiText-2 dataset (Merity et al., 2017) along with model parameters in Table 2.

**Scaling Laws.** Following Gu and Dao, we demonstrate Chinchilla scaling (Hoffmann et al., 2022) for our architectures by training on a compute-optimal budget of 20 tokens per parameter. Our models exhibit scaling behavior similar to the baseline architecture, as illustrated in Figure 10.

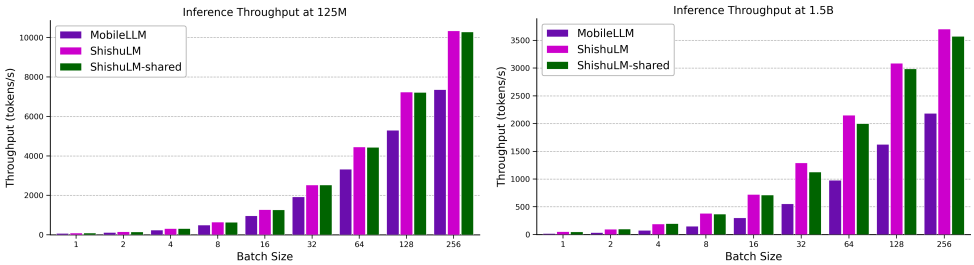
## 5 Throughput and Efficiency

We quantify the efficiency gains achieved by SHISHULM in this section. Since self-attention incurs $\mathcal{O}(T^2)$ computational complexity with respect to sequence length $T$ and MLP operations are $\mathcal{O}(T)$, reducing the number of attention layers leads to tangible throughput improvements. In SHISHULM, we remove approximately $1/3$ of full decoder layers and replace them with MLP-only blocks. As the number of added MLP layers is




(a) 125M (b) 1.5B

Figure 8: Generation latency (ms) of models at the (a) 125M and (b) 1.5B scale, with the number of new tokens generated. Prefill Cache size is set to 2048 tokens.




(a) 125M (b) 1.5B

Figure 9: Inference throughput (tokens/s) of models at the (a) 125M and (b) 1.5B scale, with different batch sizes. Prefill Cache is set to 2048 tokens and the models are allowed to generate 128 tokens.

| **Model** | **# Params** | **PPL(↓)** |
|---|---|---|
| MobileLLM-125 | 124.6M | 38.35 |
| SHISHULM-125 | 123.7M | 38.32 |
| SHISHULM-125-S | 107.8M | 39.22 |
| MobileLLM-350 | 343.3M | 30.59 |
| SHISHULM-350 | 342.8M | 30.88 |
| SHISHULM-350-S | 288.8M | 32.47 |
| MobileLLM-600 | 603M | 27.92 |
| SHISHULM-600 | 603M | 28.14 |
| SHISHULM-600-S | 497M | 29.50 |
| MobileLLM-1B | 1B | 25.60 |
| SHISHULM-1B | 1B | 25.65 |
| SHISHULM-1B-S | 849M | 26.43 |
| MobileLLM-1.5B | 1.55B | 24.49 |
| SHISHULM-1.5B | 1.55B | 24.42 |
| SHISHULM-1.5B-S | 1.3B | 24.68 |

Table 2: Wikitext perplexity of various models. Lower is better.

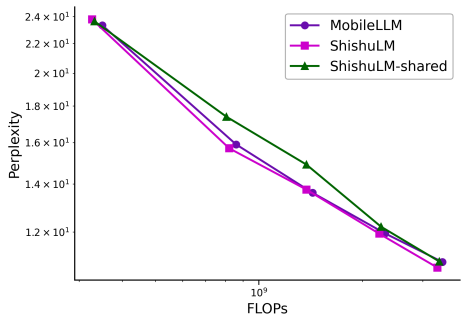



Figure 10: Scaling Laws for our models.

fewer than the number of removed attention sub-layers (approximately 5–7 MLP layers added in place of 10–18 attention sub-layers removed), improvements are seen both at training and inference time.

**Training.** We plot the a single forward pass and a combined forward-backward pass (as relevant for training) across various sequence lengths ranging from 64 to 4096. The results for the 125M and 1.5B scale are shown in Figures 6 and 7. Detailed measurement results are reported in Tables 6 and 7.

| Model | HellaSwag | PIQA | Arc-E | Arc-C | WinoGrande | OBQA | SIQA | BoolQ | Avg |
|---|---|---|---|---|---|---|---|---|---|
| MobileLLM-125M | **30.30** | 60.66 | 36.28 | 20.99 | 51.62 | 26.60 | **36.64** | 37.58 | 37.58 |
| SHISHULM-125M | 29.83 | **61.37** | 36.07 | 22.87 | **52.80** | **28.60** | 36.49 | **62.08** | **41.26** |
| SHISHULM-125M-S | 29.64 | 60.01 | **36.57** | **22.95** | 50.83 | 27.80 | **36.64** | 61.35 | 40.72 |
| MobileLLM-350M | 33.06 | **63.66** | 38.22 | 22.27 | 51.14 | 26.60 | 36.80 | **62.08** | 41.73 |
| SHISHULM-350M | **33.10** | **63.66** | **38.80** | **22.70** | **54.30** | 29.00 | **37.46** | 60.76 | **42.47** |
| SHISHULM-350M-S | 32.12 | 62.13 | 37.96 | 22.18 | 50.12 | **29.20** | 37.05 | 62.02 | 41.60 |
| MobileLLM-600M | **35.07** | 63.71 | 39.02 | 23.38 | 51.62 | **30.00** | 37.36 | **61.35** | 42.69 |
| SHISHULM-600M | 34.52 | **63.93** | **40.28** | 23.29 | **52.57** | 29.60 | 37.56 | 61.01 | **42.84** |
| SHISHULM-600M-S | 33.15 | 63.17 | 38.64 | **25.00** | 51.38 | 29.80 | **38.08** | 60.89 | 42.51 |
| MobileLLM-1B | 36.49 | 66.54 | **41.12** | **24.40** | 50.36 | 28.80 | 37.10 | 56.94 | 42.72 |
| SHISHULM-1B | **36.59** | **66.76** | **41.12** | 23.55 | 50.28 | 29.60 | 37.41 | 61.35 | 43.33 |
| SHISHULM-1B-S | 36.01 | 65.67 | 40.82 | 22.70 | **52.17** | **30.00** | **37.92** | **61.71** | **43.38** |
| MobileLLM-1.5B | 37.57 | **66.00** | 41.12 | 23.46 | 51.22 | 28.60 | 37.26 | **61.19** | 43.30 |
| SHISHLM-1.5B | **38.29** | 65.94 | **42.42** | **25.09** | 51.62 | 29.60 | 37.21 | 61.04 | **43.90** |
| SHISHULM-1.5B-S | 37.62 | 65.51 | 41.84 | 24.23 | **51.85** | **29.80** | **38.13** | 59.63 | 43.58 |

Table 3: Zero-shot evaluation results.

**Inference.** We plot the decrease in generation latency in Figure 8 and increase in inference throughput in Figure 9 at the 125M and 1.5B scale. Detailed measurements along can be found in Tables 8 and 10. Memory reduction due to reduction in KV Cache (and reduced parameters in shared versions) are shown in Table 9. We note that all efficiency improvements reported here are *orthogonal* to complementary optimization techniques such as quantization and efficient attention implementation, and can be coupled with our work.

## 6 Conclusion

We proposed SHISHULM, an efficient pre-training architecture for decoder-only language models, motivated by depth-wise redundancy in attention sublayers and the staged nature of information processing in transformers. We demonstrated that replacing the top-layer full decoder blocks with a suitable number of MLP-only blocks, while approximately preserving the total parameter count yields pre-trained models that perform comparably to full-attention baselines across scales from 125M to 1.5B parameters, while achieving meaningful improvements in training and inference throughput. Building on the high distributional similarity between adjacent MLP-only layers, we further introduced a weight-shared variant of SHISHULM that reduces parameter count with minimal performance degradation.

Our findings are directly grounded in and consistent with interpretability research showing that (i) information integration across tokens is largely complete by the midpoint of the network, (ii) later layers primarily refine the output probability distribution in a manner that does not require cross-token attention, and (iii) adjacent layers perform highly similar computations. We believe our work opens directions for future research into parameter-efficient language model architectures that are grounded in an understanding of how transformers use their computational capacity.

## 7 Limitations

First, all pre-training experiments are conducted on models up to 1.5B parameters. Second, our experiments are restricted to the LLaMA architectural family. Future work will extend these strategies to other model families such as Qwen (Bai et al., 2023), Gemma (Team et al., 2024), Pythia (Biderman et al., 2023), etc., to assess generality. Third, the optimal fraction of retained attention layers (approximately $2/3$ in our setting) may vary with architecture depth, width, or data regime, and a principled approach to determining this ratio would be valuable. Finally, it would be interesting to investigate the reasoning capabilities of post-trained SHISHULM models, and their robustness to known adversarial and privacy attacks in the AI security literature.

## References


Amit Ben Artzy and Roy Schwartz. 2024. Attend first, consolidate later: On the importance of attention in different LLM layers. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 177–184, Miami, Florida, US. Association for Computational Linguistics.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Federico Barbero, Alvaro Arroyo, Xiangming Gu, Christos Perivolaropoulos, Petar Veličković, Razvan Pascanu, and Michael M. Bronstein. 2025. Why do LLMs attend to the first token? In *Second Conference on Language Modeling*.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2023. Quantizable transformers: Removing outliers by helping attention heads do nothing. In *Advances in Neural Information Processing Systems*, volume 36, pages 75067–75096. Curran Associates, Inc.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. 2024. Reducing transformer key-value cache size with cross-layer attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D Manning. 2024. MoEUT: Mixture-of-experts universal transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Róbert Csordás, Christopher D Manning, and Christopher Potts. 2025. Do language models use their depth efficiently? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*.

Enrique Queipo de Llano, Alvaro Arroyo, Federico Barbero, Xiaowen Dong, Michael M. Bronstein, Yann LeCun, and Ravid Shwartz-Ziv. 2026. Attention sinks and compression valleys in LLMs are two sides of the same coin. In *The Fourteenth International Conference on Learning Representations*.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11. Special issue on deep reinforcement learning.

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

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, and 5 others. 2023. A framework for few-shot language model evaluation.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. 2025. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, and 1 others. 2023. Phi-2: The surprising power of small language models. *Microsoft Research Blog*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Jerome Ku, Eric Nguyen, David W. Romero, Garyk Brixi, Brandon Yang, Anton Vorontsov, Ali Taghibakhshi, Amy X Lu, Dave P. Burke, Greg Brockman, Stefano Massaroli, Christopher R'e, Patrick D. Hsu, Brian L Hie, Stefano Ermon, and Michael Poli. 2025. Systems and algorithms for convolutional multi-hybrid language models at scale. *ArXiv*, abs/2503.01868.

Vedang Lad, Wes Gurnee, and Max Tegmark. 2024. The remarkable robustness of LLMs: Stages of inference? In *ICML 2024 Workshop on Mechanistic Interpretability*.

Barak Lenz, Opher Lieber, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, Daniel Gissin, Daniel Jannai, Dor Muhlgay, Dor Zimberg, Edden M. Gerber, Elad Dolev, Eran Krakovsky, Erez Safahi, Erez Schwartz, and 42 others. 2025. Jamba: Hybrid transformer-mamba language models. In *The Thirteenth International Conference on Learning Representations*.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. 2024. Mobilellm: optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 21702–21720. Curran Associates, Inc.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *Preprint*, arXiv:2403.03853.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

Shashank Rajput, Ying Sheng, Sean Owen, and Vitaliy Chiley. 2024. Inference-friendly models with MixAttention. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pages 370–381. PMLR.

Y. Rubner, C. Tomasi, and L.J. Guibas. 1998. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 59–66.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

Shoaib Ahmed Siddiqui, Xin Dong, Greg Heinrich, Thomas Breuel, Jan Kautz, David Krueger, and Pavlo Molchanov. 2024. A deeper look at depth pruning of LLMs. In *ICML 2024 Workshop on Theoretical Foundations of Foundation Models*.

Nikolai V Smirnov. 1939. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bulletin Mathématique de l'Université de Moscou*, 2(2):3–14.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. 2024. Sleb: streamlining llms through redundancy verification and elimination of transformer blocks. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024a. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024b. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 7339–7361. Curran Associates, Inc.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, and 89 others. 2024. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Haoyi Wu and Kewei Tu. 2024. 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)*, pages 11175–11188, Bangkok, Thailand. Association for Computational Linguistics.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024. LaCo: Large language model pruning via layer collapse. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6401–6417, Miami, Florida, USA. Association for Computational Linguistics.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

# A Related Work

## A.1 Redundancy in LLMs

Previous work in the field of LLM pruning has identified several redundancies in pre-trained models. At the weight level, *Structured* pruning involves to removing entire rows, columns, or blocks from weight matrices (Ashkboos et al., 2024; Ma et al., 2023), while *unstructured* pruning sets individual weights to zero, yielding sparse matrices (Frantar and Alistarh, 2023; Sun et al., 2024a). Block-level pruning eliminates entire decoder layers by quantifying their importance. Men et al. (2024) quantify layer importance using the expected cosine similarity between block inputs and outputs, Gromov et al. (2025) similarly use cosine similarity to remove contiguous blocks of layers, followed by recovery fine-tuning via QLoRA. Yang et al. (2024) replace multiple consecutive layers with a single layer by merging their parameters. Siddiqui et al. (2024) propose a Shapley value-based metric over

a calibration dataset, while Song et al. (2024) iteratively eliminate the most redundant block based on calibration perplexity. A common finding across these works is that later layers and in particular later-layer attention sub-layers tend to exhibit the highest redundancy. However, all pruning methods continue to bear the full costs of pre-training the original over-parameterised model, while additionally incurring redundancy identification and post-training recovery overheads.

Seeking to reduce KV cache requirements, several works have attempted to share the key-value pairs across attention layers. Brandon et al. proposed sharing KV pairs across adjacent layers in groups of two, reducing the KV cache size by roughly 50%. Sun et al. also compress the KV cache by 50% by computing KV pairs only for the first half of the model and sharing them for the remaining layers. Rajput et al. place sliding window attention layers sandwiched between global attention layers, sharing keys and values across the global layers and between consecutive sliding window layers. Wu and Tu further compress the KV cache by computing keys and values only in the top layers and pairing them with queries in the bottom layers.

### A.2 Sub-quadratic Sequence Models and Hybrid Architectures

A separate line of research attempts to replace the $\mathcal{O}(T^2)$ self-attention mechanism with sub-quadratic alternatives. Linear attention methods (Katharopoulos et al., 2020) reformulate attention using kernel approximations to achieve linear-time computation, State space models (SSMs) such as Mamba (Gu and Dao, 2024) sidestep the attention mechanism entirely, and rather than treating all input tokens equally, Mamba dynamically determines which information to retain in its bounded hidden state to achieve $O(1)$ space complexity and $O(T)$ time complexity.

However, these approaches require specialized hardware kernels and benefits are typically obtained at very high context lengths. Hybrid approaches such as models (Lenz et al., 2025; Ku et al., 2025) interleave attention layers with SSM or local convolution layers, seeking to combine the strengths of both paradigms.

## B Metrics for measuring similarity in adjacent layer weight distribution

We measure the Earth Mover's distance and KS-statistic between the discrete probability distributions of weights in the gate, up and down projection matrices. While the Earth Mover's Distance captures the overall effort required to transform one distribution into another, the KS-statistic captures the maximum point-wise deviation between the distributions. Calculations using both metrics suggest that distributions learned in adjacent layers are very similar.

The formal definitions of these metrics are as follows:

**Earth Mover's Distance (EMD)** Earth Mover's Distance (Rubner et al., 1998) (EMD), also known as the Wasserstein distance, is a metric for quantifying the dissimilarity between probability distributions by framing the comparison as an optimal transport problem. Given two distributions, EMD computes the minimum cost required to transform one distribution into another, where the cost is determined by the amount of *probability mass* that must be moved and the distance over which it must be transported.

In the univariate case, this reduces to computing the integral of the absolute difference of the Cumulative Distribution Functions. Formally, for two univariate distributions $A$ and $B$ with CDFs $F_A(x)$ and $F_B(x)$ respectively, we have

$$EMD(A, B) = \int_{-\infty}^{\infty} |F_A(x) - F_B(x)| \, dx.$$

Since EMD is sensitive to scale, we instead use normalized EMD. For distribution $P_i$ at layer $i$, we have

$$EMD_{norm}(P_i, P_j) = \frac{EMD(P_i, P_j)}{(AAD_i + AAD_j)/2}$$

where

$$AAD_i = \mathbb{E}\left(|P_i - \mathbb{E}(P_i)|\right)$$

is the average absolute deviation from the mean.

**Kolmogorov-Smirnov Statistic** The two-sample Kolmogorov-Smirnov statistic, introduced by (Smirnov, 1939), is defined as follows. Let $F_A$ and $F_B$ denote two empirical cumulative distribution functions (ECDFs); then

$$KS(A, B) = \sup_{x \in \mathbb{R}} |F_A(x) - F_B(x)|$$

where $D \in [0, 1]$, with $D = 0$ indicating identical distributions and $D = 1$ indicating complete separation.

## C Training and Architecture details

In this section we describe the details of our pretraining experiments. Our implementation utilizes HuggingFace Transformers (Wolf et al., 2020) and kernel replacement with FlashAttention 2 (Dao, 2024). For all our models, we use the Silu activation function (Elfwing et al., 2018) along with rotary position embeddings (RoPE) (Su et al., 2024). We use the AdamW optimizer (Loshchilov and Hutter, 2019) with

- $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- warmup ratio = $0.05$
- weight decay $= 1 \times 10^{-2}$
- dropout $= 0$

The learning rates use a cosine scheduler (Loshchilov and Hutter, 2017). We use gradient clipping with maximum norm $1.0$. RoPE epsilon value is set to $10^{-5}$. The hyperparameters used for the parent models are the same as those for the corresponding derived ones and are described in Table 4.

We use 5, 10 and 50 B token subsets of the SlimPajama dataset (Soboleva et al., 2023) and tokenize using the MobileLLM tokenizers. The block size (sequence length) of a training sample is set to 2048 tokens. Validation perplexity is measured on a hold out set of 2000 sequences of 2048 ($\sim 4$ million tokens).

All pretraining experiments are done on 8 NVIDIA H200 GPUs with 144 GB VRAM. We use mixed precision training with bfloat16. The batch sizes are adjusted based on the model size.

## D Loss Curves

The plots of training loss against steps is shown in Figure 13 for all the models.

## E Training Speedup

We measure the forward latency in Table 6 and combined forward + backward latency in 7 for all the models across sequence lengths varying from 64 to 4096.

## F Inference speedup and memory

To measure improvements during inference, we set the prefill context to 2048 tokens and measure the time taken to generate 64 to 2048 new tokens in Table 8 and the peak GPU memory consumption in 9. Following (Gu and Dao, 2024) we measure the generation throughput by setting the prefill context to 2048 and allowing models to generate 128 tokens for various batch sizes. The throughput is calculated using the formula:

$$\text{Throughput} = \frac{127}{\text{Total time} - \text{TTFT}} \qquad (1)$$

where TTFT (Time To First Token) represents the time taken to generate the first token. The difference between Total time to generate 128 tokens and TTFT gives the time taken to generate 127 tokens. The results are shown in Table 10.

Please note that in all these experiments, the $< eos >$ token is set to `None`, so that the model does not stop generating before the intended number of tokens.

## G AI assistant usage

Claude Sonnet 4.6 was used to generate the codes for generating plots and measuring statistics.

| Scale | Batch-size | Gradient-accumulation | Initial lr |
|---|---|---|---|
| 125M | 64 | 1 | $2 \times 10^{-3}$ |
| 350M | 64 | 1 | $2 \times 10^{-3}$ |
| 600M | 32 | 2 | $1.5 \times 10^{-3}$ |
| 1B | 24 | 3 | $1 \times 10^{-3}$ |
| 1.5B | 16 | 3 | $1 \times 10^{-3}$ |

Table 4: Hyperparameters using across different scales - per GPU batch size, number of gradient accumulation steps and the initial learning rate.

| Model | Layer Config | #Head | #KV-Head | Emb Dim | Hidden Dim | #Params |
|---|---|---|---|---|---|---|
| MobileLLM-125M | 30 + 0 | 9 | 3 | 576 | 1536 | 124,635,456 |
| SHISHULM-125M | 20 + 13 | 9 | 3 | 576 | 1536 | 123,746,688 |
| SHISHULM-125M-shared | 20 + 7*2 | 9 | 3 | 576 | 1536 | 107,817,984 |
| MobileLLM-350M | 32 + 0 | 15 | 5 | 960 | 2560 | 345,355,200 |
| SHISHULM-350M | 22 + 13 | 15 | 5 | 960 | 2560 | 342,890,880 |
| SHISHULM-350M-shared | 22 + 6*2 | 15 | 5 | 960 | 2560 | 288,816,000 |
| MobileLLM-600M | 40 + 0 | 18 | 6 | 1152 | 3072 | 603,188,352 |
| SHISHULM-600M | 25 + 20 | 18 | 6 | 1152 | 3072 | 603,176,832 |
| SHISHULM-600M-shared | 25 + 10*2 | 18 | 6 | 1152 | 3072 | 496,996,992 |
| MobileLLM-1B | 54 + 0 | 20 | 5 | 1280 | 3584 | 1,005,461,760 |
| SHISHULM-1B | 36 + 23 | 20 | 5 | 1280 | 3584 | 1,000,529,920 |
| SHISHULM-1B-shared | 36 + 12*2 | 20 | 5 | 1280 | 3584 | 849,127,680 |
| MobileLLM-1.5B | 54 + 0 | 25 | 5 | 1600 | 4352 | 1,557,844,800 |
| SHISHULM-1.5B | 36 + 23 | 25 | 5 | 1600 | 4352 | 1,556,000,000 |
| SHISHULM-1.5B-shared | 36 + 12*2 | 25 | 5 | 1600 | 4352 | 1,316,692,800 |

Table 5: Architecture details of the MobileLLM and SHISHULM variants. Layer config $m + n$ means $m$ full decoder layers followed by $n$ MLP-only layers. In the shared versions we write $m + n * 2$ to represent $n$ pairs of weight-shared MLP-layers.

| Model | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|---|---|
| MobileLLM-125 | 33.838 | 33.673 | 34.062 | 35.546 | 36.12 | 36.058 | 38.241 |
| SHISHULM-125 | 25.713 | 25.974 | 26.308 | 27.3 | 27.763 | 27.881 | 29.547 |
| SHISHULM-125-S | 26.146 | 26.27 | 26.578 | 27.663 | 28.295 | 28.29 | 30.086 |
| MobileLLM-350 | 36.675 | 36.633 | 38.57 | 38.908 | 39.724 | 40.488 | 40.611 |
| SHISHULM-350 | 28.873 | 28.811 | 30.48 | 30.863 | 31.241 | 32.213 | 32.12 |
| SHISHULM-350-S | 27.641 | 27.786 | 29.404 | 29.781 | 30.263 | 30.918 | 30.953 |
| MobileLLM-600 | 45.905 | 45.968 | 48.257 | 49.156 | 49.949 | 51.14 | 51.175 |
| SHISHULM-600 | 18.642 | 18.851 | 21.483 | 32.193 | 21.283 | 20.955 | 26.898 |
| SHISHULM-600-S | 18.183 | 18.424 | 19.481 | 19.381 | 19.689 | 20.376 | 26.53 |
| MobileLLM-1B | 42.781 | 42.835 | 46.902 | 46.38 | 62.004 | 63.51 | 63.475 |
| SHISHULM-1B | 24.867 | 25.378 | 27.359 | 27.386 | 26.977 | 27.928 | 39.69 |
| SHISHULM-1B-S | 25.525 | 25.722 | 28.098 | 27.799 | 27.732 | 28.344 | 39.96 |
| MobileLLM-1.5B | 60.877 | 64.16 | 64.563 | 46.336 | 47.475 | 59.212 | 96.727 |
| SHISHULM-1.5B | 25.423 | 27.04 | 27.401 | 27.515 | 34.946 | 49.578 | 84.803 |
| SHISHULM-1.5B-S | 25.829 | 27.05 | 27.454 | 27.87 | 35.278 | 50.354 | 85.711 |

Table 6: Forward pass latency (ms) of all the models against sequence length, averaged across 3 trials.

| Model | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|---|---|
| MobileLLM-125 | 82.191 | 82.766 | 83.57 | 86.177 | 88.54 | 89.872 | 93.201 |
| SHISHULM-125 | 63.941 | 64.497 | 65.16 | 67.67 | 69.374 | 70.842 | 73.787 |
| SHISHULM-125-S | 65.095 | 65.514 | 66.202 | 68.591 | 70.744 | 71.975 | 75.071 |
| MobileLLM-350 | 89.138 | 88.814 | 91.886 | 94.526 | 97.127 | 98.614 | 99.329 |
| SHISHULM-350 | 71.373 | 70.759 | 74.009 | 76.056 | 78.287 | 79.633 | 79.606 |
| SHISHULM-350-S | 67.054 | 68.878 | 71.662 | 73.969 | 75.975 | 77.478 | 77.044 |
| MobileLLM-600 | 110.907 | 112.194 | 116.174 | 118.92 | 121.432 | 125.012 | 125.321 |
| SHISHULM-600 | 45.937 | 46.175 | 78.142 | 74.789 | 49.911 | 52.741 | 80.529 |
| SHISHULM-600-S | 44.739 | 44.94 | 47.058 | 47.433 | 48.373 | 50.521 | 79.434 |
| MobileLLM-1B | 108.693 | 109.072 | 114.121 | 113.433 | 150.873 | 155.192 | 159.653 |
| SHISHULM-1B | 64.178 | 64.591 | 67.812 | 67.319 | 68.135 | 74.648 | 121.447 |
| SHISHULM-1B-S | 63.979 | 64.152 | 67.462 | 67.272 | 67.958 | 75.196 | 122.55 |
| MobileLLM-1.5B | 151.583 | 155.057 | 114.899 | 112.472 | 116.32 | 166.525 | 283.634 |
| SHISHULM-1.5B | 64.003 | 65.493 | 66.07 | 64.616 | 91.482 | 143.921 | 259.04 |
| SHISHULM-1.5B-S | 63.672 | 65.448 | 65.949 | 64.901 | 92.488 | 145.469 | 261.981 |

Table 7: Forward + backward latency (ms) of all the models against sequence length, averaged across 3 trials.

| Model | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|
| MobileLLM-125 | 1862.351 | 3676.282 | 7387.964 | 14800.752 | 21479.821 | 59027.564 |
| SHISHULM-125 | 1405.994 | 2801.33 | 5572.858 | 11140.375 | 22453.405 | 44806.15 |
| SHISHULM-125-S | 1407.675 | 2839.292 | 5686.073 | 11339.453 | 22561.019 | 45125.332 |
| MobileLLM-350 | 1975.222 | 3946.171 | 7851.299 | 15700.092 | 26234.292 | 62383.566 |
| SHISHULM-350 | 1531.328 | 3064.831 | 6066.522 | 12101.677 | 24315.97 | 48608.603 |
| SHISHULM-350-S | 1481.14 | 2953.442 | 5919.166 | 11772.223 | 23543.782 | 47084.087 |
| MobileLLM-600 | 2409.119 | 4357.069 | 9851.366 | 19632.83 | 39126.498 | 78450.316 |
| SHISHULM-600 | 912.705 | 1829.799 | 3664.875 | 7503.226 | 14672.568 | 32923.355 |
| SHISHULM-600-S | 905.391 | 1809.028 | 3587.289 | 7179.725 | 14313.755 | 28637.825 |
| MobileLLM-1B | 2584.45 | 5127.022 | 9900.442 | 20585.952 | 52403.289 | 83281.498 |
| SHISHULM-1B | 1260.264 | 2514.476 | 5012.187 | 10003.189 | 19964.959 | 39916.396 |
| SHISHULM-1B-S | 1260.422 | 2519.033 | 5015.874 | 10008.488 | 19985.736 | 39907.113 |
| MobileLLM-1.5B | 2235.186 | 4350.952 | 9502.356 | 19049.303 | 37895.821 | 92460.617 |
| SHISHULM-1.5B | 1281.17 | 2523.257 | 5006.245 | 9991.8 | 19934.048 | 39833.26 |
| SHISHULM-1.5B-S | 1289.384 | 2542.772 | 5052.844 | 10084.823 | 20065.022 | 40217.857 |

Table 8: Generation latency (ms) with number of tokens generated, averaged across 3 trials. Prefill cache size is set to 2048 tokens.

| Model | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|
| MobileLLM-125M | 379.18 | 379.18 | 379.18 | 379.18 | 401.95 | 401.95 |
| SHISHULM-125M | 362.48 | 362.48 | 362.48 | 362.48 | 365.06 | 368.81 |
| SHISHULM-125M-S | 332.41 | 332.41 | 332.41 | 332.41 | 338.04 | 338.04 |
| MobileLLM-350M | 890.22 | 890.22 | 890.22 | 890.22 | 892.04 | 926.92 |
| SHISHULM-350M | 854.81 | 854.81 | 854.81 | 854.81 | 854.81 | 866.66 |
| SHISHULM-350M-S | 751.05 | 751.05 | 751.05 | 751.05 | 751.05 | 759.45 |
| MobileLLM-600M | 1437.80 | 1437.80 | 1437.80 | 1437.80 | 1438.28 | 1506.06 |
| SHISHULM-600M | 1390.77 | 1390.77 | 1390.77 | 1390.77 | 1390.77 | 1419.29 |
| SHISHULM-600M-S | 1188.24 | 1188.24 | 1188.24 | 1188.24 | 1188.24 | 1216.76 |
| MobileLLM-1B | 2249.00 | 2249.00 | 2249.00 | 2270.00 | 2272.51 | 2300.64 |
| SHISHULM-1B | 2186.71 | 2186.71 | 2186.71 | 2186.71 | 2186.71 | 2201.23 |
| SHISHULM-1B-S | 1898.30 | 1898.30 | 1898.30 | 1898.30 | 1898.30 | 1912.45 |
| MobileLLM-1.5B | 3360.15 | 3360.15 | 3360.15 | 3360.15 | 3360.15 | 3419.11 |
| SHISHULM-1.5B | 3308.44 | 3308.44 | 3308.44 | 3308.44 | 3308.44 | 3322.89 |
| SHISHULM-1.5B-S | 2846.40 | 2846.40 | 2846.40 | 2846.40 | 2846.40 | 2860.85 |

Table 9: Peak GPU memory consumption during generation (MB) with number of tokens generated. Prefill cache size is set to 2048 tokens.

| Model | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| MobileLLM-125 | 70.89 | 119.93 | 242.79 | 493.60 | 966.65 | 1929.95 | 3326.93 | 5305.26 | 7360.57 |
| SHISHULM-125 | 91.25 | 155.71 | 315.61 | 645.06 | 1276.00 | 2527.69 | 4458.25 | 7243.68 | 10343.81 |
| SHISHULM-125-S | 90.95 | 154.13 | 315.90 | 642.64 | 1270.39 | 2531.76 | 4434.59 | 7216.79 | 10290.46 |
| MobileLLM-350 | 66.86 | 124.17 | 249.43 | 486.62 | 913.24 | 1615.05 | 2632.69 | 3810.32 | 4565.97 |
| SHISHULM-350 | 85.31 | 158.39 | 316.64 | 624.09 | 1174.62 | 2113.64 | 3494.86 | 5254.10 | 6371.47 |
| SHISHULM-350-S | 88.06 | 164.11 | 326.34 | 642.24 | 1211.64 | 2187.68 | 3634.63 | 5385.78 | 6634.17 |
| MobileLLM-600 | 53.48 | 99.47 | 198.56 | 385.69 | 705.05 | 1231.44 | 1945.67 | 2740.30 | 3125.01 |
| SHISHULM-600 | 72.53 | 135.97 | 274.44 | 533.06 | 977.65 | 1741.91 | 2794.68 | 4075.20 | 4787.77 |
| SHISHULM-600-S | 70.82 | 133.29 | 263.13 | 524.98 | 955.88 | 1709.45 | 2749.60 | 4016.67 | 4790.31 |
| MobileLLM-1B | 40.05 | 54.23 | 77.12 | 152.43 | 309.97 | 601.52 | 1079.43 | 1811.72 | 2525.73 |
| SHISHULM-1B | 54.73 | 99.16 | 197.43 | 385.69 | 740.87 | 1333.58 | 2163.78 | 3244.12 | 3887.21 |
| SHISHULM-1B-S | 52.75 | 98.32 | 195.20 | 387.75 | 720.57 | 1300.79 | 2162.05 | 3204.29 | 3883.05 |
| MobileLLM-1.5B | 19.57 | 37.88 | 75.87 | 150.86 | 302.83 | 557.00 | 978.34 | 1627.16 | 2186.43 |
| SHISHULM-1.5B | 52.81 | 96.33 | 192.79 | 382.35 | 724.43 | 1293.66 | 2152.20 | 3088.31 | 3704.59 |
| SHISHULM-1.5B-S | 50.06 | 100.14 | 199.95 | 370.32 | 714.35 | 1128.48 | 2000.32 | 2987.64 | 3574.30 |

Table 10: Generation throughput (tokens/s) of models across different batch sizes. Prefill cache size is set to 2048 tokens and the models are allowed to generate 128 tokens.

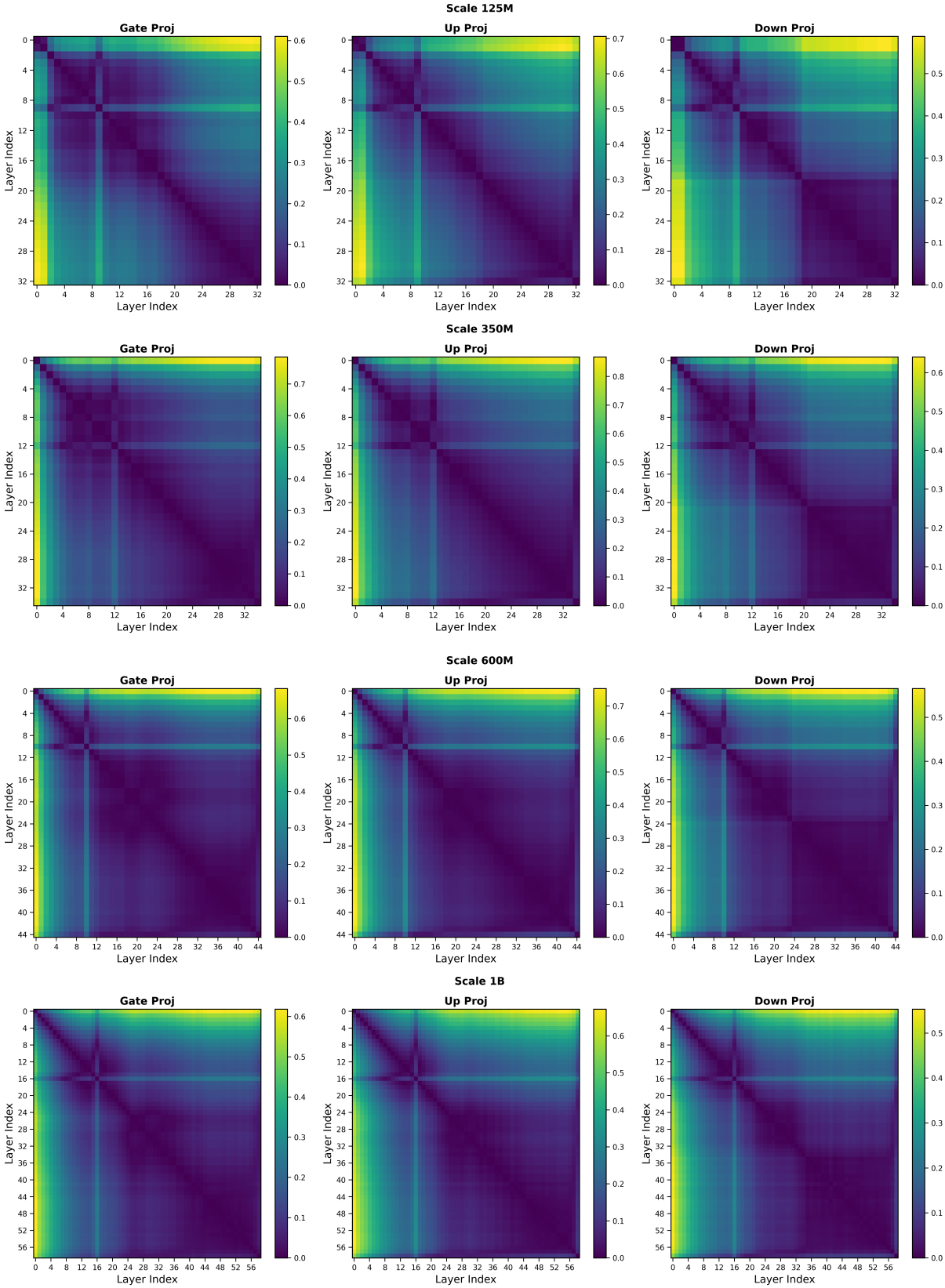



Figure 11: Weight distribution similarity using EMD across all model scales in SHISHULM.

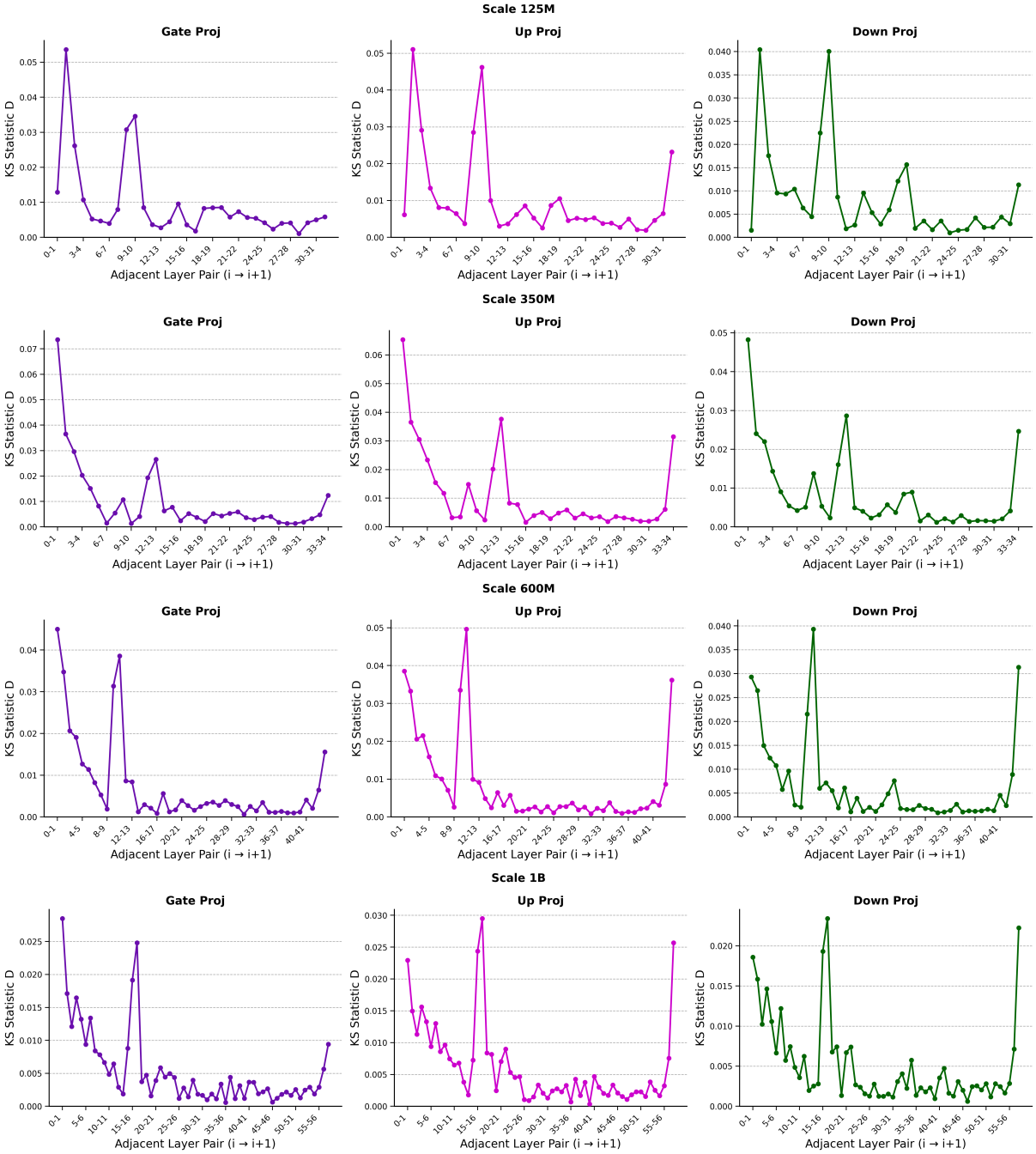



Figure 12: KS-similarity between adjacent layer distributions across all model scales in SHISHULM.

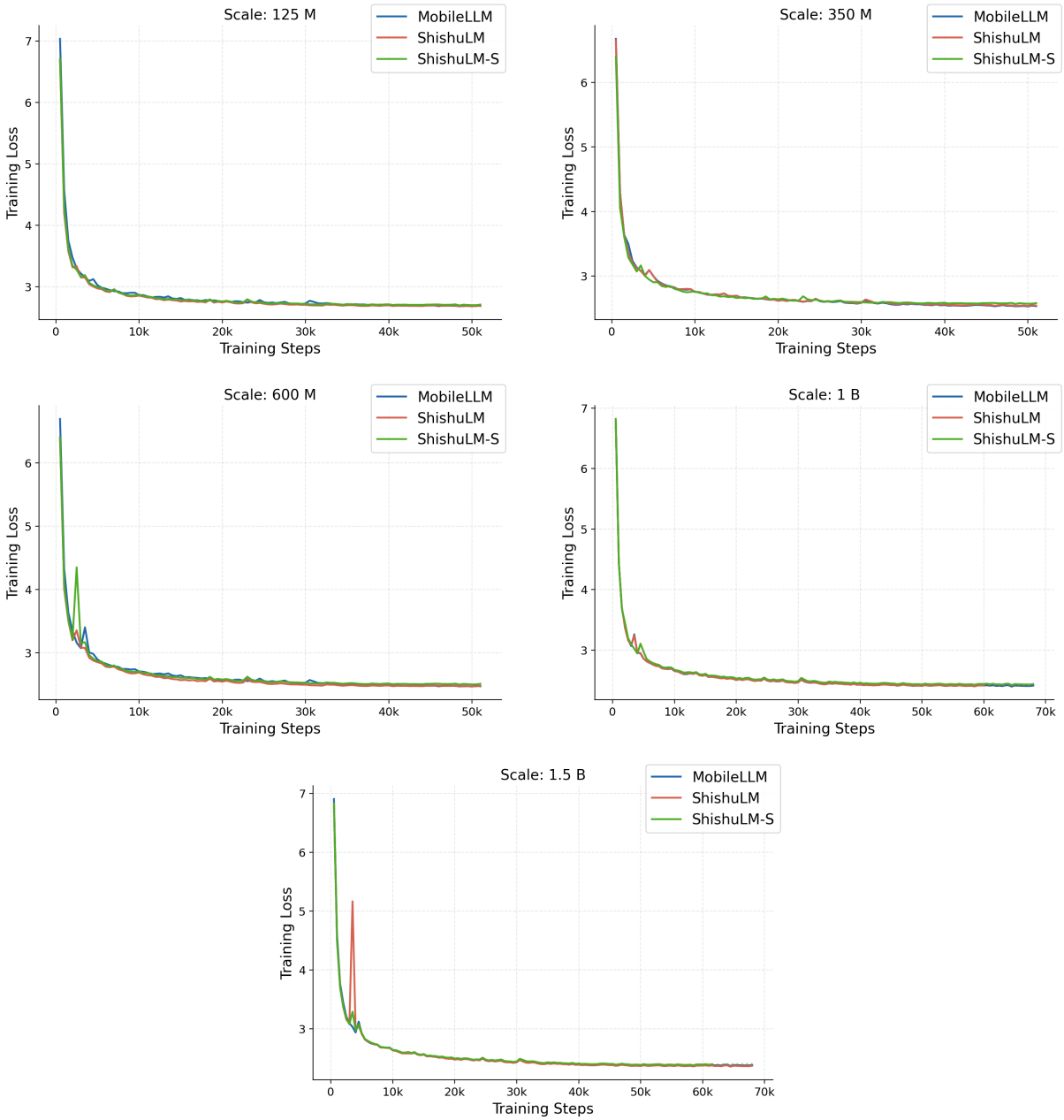



Figure 13: Training loss curves for the five model scales.