LINEAR-MOE: LINEAR SEQUENCE MODELING MEETS MIXTURE-OF-EXPERTS

Weigao Sun[1†] **Disen Lan**[1,2] **Tong Zhu**[3] **Xiaoye Qu**[1] **Yu Cheng**[4]

[1]Shanghai AI Laboratory [2]South China University of Technology
[3]Soochow University [4]The Chinese University of Hong Kong

ABSTRACT

Linear Sequence Modeling (LSM) and Mixture-of-Experts (MoEs) have recently emerged as effective architectural improvements. In this paper, we introduce Linear-MoE, a production-level system for modeling and training large-scale models that integrate LSM with MoEs. Linear-MoE leverages the advantages of both LSM modules for linear-complexity sequence modeling and MoE layers for sparsely activation, aiming to offer high performance with efficient training and deployment. The Linear-MoE system comprises two primary subsystems: Modeling and Training. The Modeling subsystem provides a unified framework supporting multiple types of LSM methods, including linear attention, SSM, and linear RNN. The Training subsystem facilitates efficient training by incorporating advanced parallelism techniques like Tensor, Pipeline, and Expert Parallelism, along with LASP-based Sequence Parallelism for managing very-long input sequences. The system is designed to be extensible for integrating more sequence modeling and training abilities in the future. Additionally, we explore hybrid Linear-MoE models that combine Linear-MoE layers with standard Transformer-MoE layers to further enhance model flexibility and performance. Experimental evaluations on two model series, A0.3B-2B and A1B-7B, demonstrate that Linear-MoE achieves efficiency gains while maintaining competitive performance on various benchmarks. The code is released at: `https://github.com/OpenSparseLLMs/Linear-MoE`.

1 INTRODUCTION

Most advances on Mixture-of-Experts (MoEs) studies primarily concentrate on modifying the routing mechanism or expert layers, while typically keeping the attention layers unchanged. These attention layers commonly rely on the softmax self-attention mechanism introduced in the Transformer architecture (Vaswani et al., 2017). The softmax-based self-attention has proven to be highly effective for sequence modeling tasks across various data types. However, a significant limitation of this mechanism is its computational complexity, which grows quadratically with the input sequence length. This complexity can lead to substantial computational costs, especially during training, making it a challenge for models that need to handle long sequences efficiently.

Linear sequence modeling (LSM) has recently gained significant attention due to its impressive efficiency in both training and inference. These methods function similarly to recurrent neural networks (RNNs) with matrix-valued hidden states, allowing them to achieve linear-time training and constant-memory inference. This efficiency is largely due to the fact that LSM techniques bypass the computation of attention scores and eliminate the need for maintaining a key-value (KV) cache. There are three primary approaches to linear sequence modeling: linear attention (Katharopoulos et al., 2020), state space modeling (SSM) (Gu & Dao, 2023; Dao & Gu, 2024), and linear RNN (Peng et al., 2023; Qin et al., 2024d). Linear attention is a variation of the traditional softmax attention mechanism, replacing the exponential kernel with a simpler dot product between key and query vectors, which enables the use of the right-product kernel trick to reduce computational complexity.

† Project lead. Corresponding to Weigao Sun (sunweigao@outlook.com). Work done during Disen Lan's internship at Shanghai AI Laboratory.

SSM approaches, such as Mamba and Mamba2, stem from control theory and represent sequence modeling as dynamic systems. Meanwhile, linear RNN methods address the limitations of traditional RNNs in modeling long contexts by enabling parallel training of RNN models. These different methods, linear attention, SSM, and linear RNN, share a common mathematical foundation and exhibit similar performance on sequence modeling tasks Dao & Gu (2024); Peng et al. (2024); Qin et al. (2024b); Yang et al. (2024b). In fact, they all employ a unified recurrence framework expressed as $\mathbf{M}_s = \mathbf{M}_{s-1} + \widehat{\mathbf{M}}_s$, where $\mathbf{M}_s$ denotes the memory state and $\widehat{\mathbf{M}}_s$ represents the incremental memory update at the s-th token.

In this paper, we introduce Linear-MoE, a production-level system designed for modeling and training of large-scale MoE models with LSM modules integrated. The Linear-MoE system is composed of two key subsystems: Modeling and Training. The Modeling subsystem provides a unified modeling framework for Linear-MoE models, which combine LSM modules with MoE layers. It supports three main types of LSM methods: linear attention, SSM, and linear RNN. For each type, multiple method instances are implemented under a unified formulation. While the Training subsystem is designed to achieve efficient training of Linear-MoE models on modern accelerators. In addition to supporting state-of-the-art training techniques such as Tensor Parallelism (TP), Pipeline Parallelism (PP), and Expert Parallelism (EP), we incorporate LASP, a specialized Sequence Parallelism (SP) technique for LSM modules, which is particularly effective for handling extremely long input sequences on Linear-MoE architecture. Importantly, the system is designed to be extensible, enables more advanced sequence modeling methods or training techniques integrated in the future.

Furthermore, the system is not limited to pure Linear-MoE models, where every layer utilizes LSM modules. We also explore efficient modeling and training for hybrid Linear-MoE models, which combine Linear-MoE layers with standard Transformer-MoE layers. For these hybrid models, we introduce an SP method that employs distinct computational and communication strategies tailored to the different types of layers. In the experiments, we introduce two series of Linear-MoE models: A0.3B-2B (with a total of 2 billion parameters and 0.3 billion activated) and A1B-7B (with a total of 7 billion parameters and 1 billion activated). Each series includes multiple model instances incorporating different LSM modules. We pretrain these models on the public SlimPajama corpus and evaluate their efficiency and performance across several mainstream benchmarks to assess the effectiveness of proposed Linear-MoE architecture.

2 Linear-MoE System

2.1 Modeling

2.1.1 Unified Linear Sequence Modeling

The standard softmax attention (Vaswani et al., 2017), commonly used in transformer models, whose parallel computation form during training can typically be expressed as: $\mathbf{O} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$. Here, the matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O} \in \mathcal{R}^{N \times d}$ correspond to the query, key, value, and output matrices, respectively. The matrices $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ are linear projections of the input matrix $\mathbf{X} \in \mathcal{R}^{N \times d}$, defined as $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}_K$, and $\mathbf{V} = \mathbf{X}\mathbf{W}_V$, where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathcal{R}^{d \times d}$ are learnable weight matrices. Here, N and d represent the sequence length and hidden dimension.

Linear Attention (Katharopoulos et al., 2020) as one of the representative LSM methods, has emerged as a viable alternative to traditional softmax attention by implementing two primary modifications. First, it eliminates the Softmax$(\cdot)$ operation, instead embedding it within a kernel feature map. Second, it leverages the associative property of matrix multiplication, reconfiguring $(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$ into $\mathbf{Q}(\mathbf{K}^\top\mathbf{V})$. These changes reduce both the computational and memory complexity from $O(N^2 d)$ to $O(N d^2)$. This approach is frequently referred to as the right-product kernel trick, as it prioritizes matrix multiplication on the right side.

While during inference, both softmax self-attention and linear attention handle a single token at each iteration. Given the s-th token $\mathbf{x}_s \in \mathcal{R}^{1 \times d}$, softmax self-attention computes requiring the storage of an expanding set of keys $\{k_1, \cdots, k_s\}$ and values $\{v_1, \cdots, v_s\}$ i.e., the KV cache, which leads to a significant memory burden when dealing with long input sequences:

$$\mathbf{q}_s, \mathbf{k}_s, \mathbf{v}_s = \mathbf{x}_s\mathbf{W}_Q, \mathbf{x}_s\mathbf{W}_K, \mathbf{x}_s\mathbf{W}_V, \quad \mathbf{o}_s = \frac{\sum_{i=1}^{s} \exp(\mathbf{q}_s {\mathbf{k}_i}^\top)\mathbf{v}_i}{\sum_{i=1}^{s} \exp(\mathbf{q}_s \mathbf{k}_i^\top)}. \tag{1}$$

Table 1: **Instances of Linear Sequence Modeling Methods.** All instances listed follow the unified formulation in Eq. (4). Here, $a \in \mathbb{R}$, $a_s \in \mathbb{R}$, $\mathbf{a}_s \in \mathbb{R}^d$, $\mathbf{A} \in \mathbb{R}^{d\times d}$, $\mathbf{A}_s \in \mathbb{R}^{d\times d}$ represents a fixed constant, a time-dependent scalar, a time-dependent vector, a time-independent matrix, and a time-dependent matrix, respectively. Note that the same notation may denote different variables in different instances.

LSM Method	Instance	Recurrent Update Rule	Parameter
Linear Attention	BLA	$\mathbf{M}_s = \mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	\
	Lightning Attn	$\mathbf{M}_s = a\mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	$a \in \mathbb{R}$
	RetNet	$\mathbf{M}_s = a\mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	$a \in \mathbb{R}$
	GLA	$\mathbf{M}_s = \text{diag}\{\mathbf{a}_s\}\mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	$\mathbf{a}_s \in \mathbb{R}^d$
	DeltaNet	$\mathbf{M}_s = (\mathbf{I} - a_s\mathbf{k}_s^\top \mathbf{k}_s)\mathbf{M}_{s-1} + b_s\mathbf{k}_s^\top \mathbf{v}_s$	$a_s, b_s \in \mathbb{R}$
	Rebased	$\mathbf{M}_s = \mathbf{M}_{s-1} + \phi(\mathbf{k}_s)^\top \mathbf{v}_s$	\
	GFW	$\mathbf{M}_s = \mathbf{A}_s \odot \mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	$\mathbf{A}_s \in \mathbb{R}^{d\times d}$
	GateLoop	$\mathbf{M}_s = \mathbf{A}_s \odot \mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	$\mathbf{A}_s \in \mathbb{R}^{d\times d}$
	Gated DeltaNet	$\mathbf{M}_s = a_s(\mathbf{I} - \mathbf{k}_s^\top \mathbf{k}_s)\mathbf{M}_{s-1} + b_s\mathbf{k}_s^\top \mathbf{v}_s$	$a_s, b_s \in \mathbb{R}$
	TTT	$\mathbf{M}_s = \mathbf{M}_{s-1} + b_s\nabla l(\mathbf{M}_{s-1}; \mathbf{k}_s, \mathbf{v}_s)$	$b_s \in \mathbb{R}$
	Titans	$\mathbf{M}_s = a_s\mathbf{M}_{s-1} + b_s\nabla l(\mathbf{M}_{s-1}; \mathbf{k}_s, \mathbf{v}_s)$	$a_s, b_s \in \mathbb{R}$
SSM *	S4	$\mathbf{M}_s = \exp(-(\mathbf{a}\mathbf{1}^\top)\mathbf{A}) \odot \mathbf{M}_{s-1}$ $+(\mathbf{a}\mathbf{1}^\top)\mathbf{b}^\top \mathbf{v}_s$	$\mathbf{a}, \mathbf{b} \in \mathbb{R}^d, \mathbf{A} \in \mathbb{R}^{d\times d}$
	Mamba	$\mathbf{M}_s = \exp(-(\mathbf{a}_s\mathbf{1}^\top)\mathbf{A}_s) \odot \mathbf{M}_{s-1}$ $+(\mathbf{a}_s\mathbf{1}^\top)\mathbf{k}_s^\top \mathbf{v}_s$	$\mathbf{a}_s \in \mathbb{R}^d, \mathbf{A}_s \in \mathbb{R}^{d\times d}$
	Mamba2	$\mathbf{M}_s = \exp(-ab_s) \odot \mathbf{M}_{s-1} + b_s\mathbf{k}_s^\top \mathbf{v}_s$	$a, b_s \in \mathbb{R}$
Linear RNN	HGRN2	$\mathbf{M}_s = \text{diag}\{\mathbf{a}_s\}\mathbf{M}_{s-1} + (1 - \mathbf{a}_s)^\top \mathbf{v}_s$	$\mathbf{a}_s \in \mathbb{R}^d$
	RWKV6	$\mathbf{M}_s = \text{diag}\{\mathbf{a}_s\}\mathbf{M}_{s-1} + \mathbf{k}_s^\top \mathbf{v}_s$	$\mathbf{a}_s \in \mathbb{R}^d$
	RWKV7	$\mathbf{M}_s = \text{diag}\{\mathbf{a}_s\}\mathbf{M}_{s-1} + \nabla l(\mathbf{M}_{s-1}; \mathbf{k}_s, \mathbf{v}_s)$	$\mathbf{a}_s \in \mathbb{R}^d$

* For both S4 and Mamba, the Euler Discretization (Gu et al., 2020) is applied, such that $\bar{\mathbf{B}} = \mathbf{\Delta}\mathbf{B}$, and the unprojected $\mathbf{x}_s$ is denoted as $\mathbf{v}_s$ for consistency with other formulas.

Linear attention replaces the term $\exp(\mathbf{q}_s\mathbf{k}_i^\top)$ with a kernel $k(\mathbf{x}, \mathbf{y})$ with an associated feature map ϕ, i.e., $k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$. This simplifies the calculation of $\mathbf{o}_s$ as

$$\mathbf{o}_s = \frac{\sum_{i=1}^{s} \phi(\mathbf{q}_s)\phi(\mathbf{k}_i)^\top \mathbf{v}_i}{\sum_{i=1}^{s} \phi(\mathbf{q}_s)\phi(\mathbf{k}_i)^\top}. \tag{2}$$

Letting $\mathbf{M}_s = \sum_{i=1}^{s} \phi(\mathbf{k}_i)^\top \mathbf{v}_i$ and $\mathbf{z}_s = \sum_{i=1}^{s} \phi(\mathbf{k}_i)^\top$ where $\mathbf{M}_s \in \mathcal{R}^{d\times d}$, $\mathbf{z}_s \in \mathcal{R}^{d\times 1}$, we can rewrite Eq. (2) as an RNN:

$$\mathbf{M}_s = \mathbf{M}_{s-1} + \phi(\mathbf{k}_s)^\top \mathbf{v}_s, \mathbf{z}_s \quad = \mathbf{z}_{s-1} + \phi(\mathbf{k}_s)^\top, \mathbf{o}_s = \frac{\phi(\mathbf{q}_s)\mathbf{M}_s}{\phi(\mathbf{q}_s)\mathbf{z}_s}. \tag{3}$$

Follow-up studies on SSM (e.g., Mamba2) and linear RNNs (e.g., RWKV6, HGRN2), have demonstrated their similarity with linear attention Dao & Gu (2024); Peng et al. (2024). In fact, recent studies (Qin et al., 2024b; Yang et al., 2024b) have suggested that linear attention, state space, and linear RNN sequence modeling methods can be expressed within a unified recurrence framework as:

$$\widehat{\mathbf{M}}_s = f(\mathbf{K}_s^\top, \mathbf{V}_s), \mathbf{M}_s \quad = \mathbf{\Theta}_s \diamond \mathbf{M}_{s-1} + \widehat{\mathbf{M}}_s. \tag{4}$$

In this formulation, $\widehat{\mathbf{M}}_s \in \mathcal{R}^{d\times d}$ represents the memory state corresponding to the s-th input, which is a function of $\mathbf{K}_s^\top$ and $\mathbf{V}_s$. And $\mathbf{\Theta}_s$ denotes a coefficient matrix that may be time-varying or constant (and also can be a vector or scalar). The operator "$\diamond$" can denote either standard matrix multiplication or a Hadamard product. We collect recent LSM method instances which follow the unified formulation in Eq. (4) and list them in Table 1.

2.1.2 Linear-MoE Architecture

The Linear-MoE architecture is relatively straightforward, consisting of $N\times$ stacked Linear-MoE blocks, as depicted in Fig. 1. Each Linear-MoE block includes an LSM layer and an MoE layer, with a normalization layer preceding each. The LSM layer serves as a generalized structure that

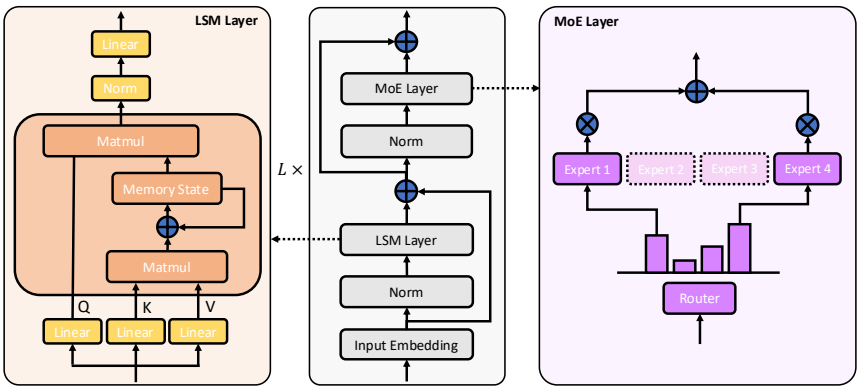

Figure 1: **Linear-MoE Architecture.** In each Linear-MoE block, there is both an LSM layer and an MoE layer, with each layer preceded by its own normalization layer. The LSM layer is designed as a flexible abstraction that integrates three types of methods: linear attention, SSM, and linear RNN, which follows a unified recurrence framework.

supports various LSM methods, specifically, linear attention, SSM, and linear RNN, each encompassing multiple instance methods. Table 1 provides an overview of these LSM method instances, unified under a common recurrence framework. This framework highlights key distinctions between instances, primarily in their handling of the prior-step memory state $\mathbf{M}_{s-1}$ and the computation of the incremental memory state $\widehat{\mathbf{M}}_s$. For the MoE layers, we retain the standard mechanisms of sparse expert activation and routing, as employed in SOTA MoE models. These mechanisms are essential for maintaining an optimal balance between model performance and computational efficiency.

Based on this prior, we propose a hybrid Linear-MoE architecture that combines Linear-MoE layers with standard (MoE) transformer layers. A practical approach for constructing these hybrid models is to periodically substitute certain Linear-MoE layers with standard MoE transformer layers within the model. For instance, in an 4-layer hybrid Linear-MoE model, denoted by "L" for Linear-MoE layers and "N" for normal transformer layers, configurations such as "LLLL" or "LNLN" may be used, depending on the desired ratio of normal transformer layers, which can be adjusted manually.

2.2 Training

2.2.1 Sequence Parallelism on Linear-MoEs

The existing methods, LASP (Sun et al., 2024b) and its modified variant LASP-2 (Sun et al., 2024a), are designed specifically to leverage the right-product-first property of LSM techniques by introducing specialized algorithms for efficient SP within LSM modules. LASP employs a point-to-point ring-style communication pattern, facilitating the exchange of incremental memory states across devices. This communication pattern is particularly effective for managing dependencies while minimizing the data transferred between devices, enhancing the scalability of SP. LASP-2 further refines this approach by replacing the ring-style communication with an all-gather collective communication operation, streamlining the entire communication process. This modification not only simplifies the communication structure but also improves the parallelism of computation and communication. By reorganizing the computation and communication flows, LASP-2 enables greater overlap between these processes, thereby boosting efficiency during both training and inference. A detailed breakdown of the LASP-2 algorithm, with and without masking, is provided in Appendix A.2.1.

In this work, we extend the capabilities of LASP-2 to the Linear-MoE system, allowing for the efficient SP training on Linear-MoE architecture, particularly when dealing with extremely long sequences across large-scale distributed clusters. This extension significantly enhances the scalability and efficiency of training Linear-MoE models with long sequences on extensive compute resources.

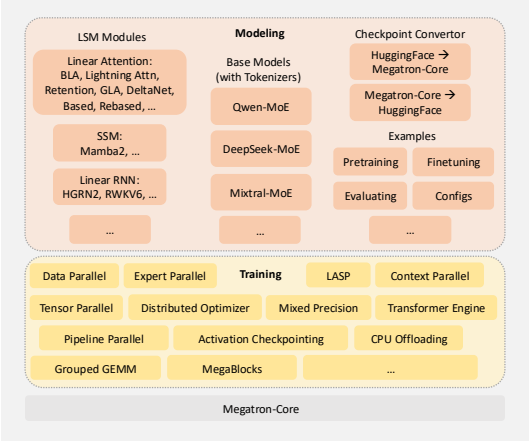

Figure 2: **Linear-MoE System Implementation.** The Linear-MoE system is composed of two main subsystems: Modeling and Training. It is developed in a non-intrusive manner on Megatron-Core.

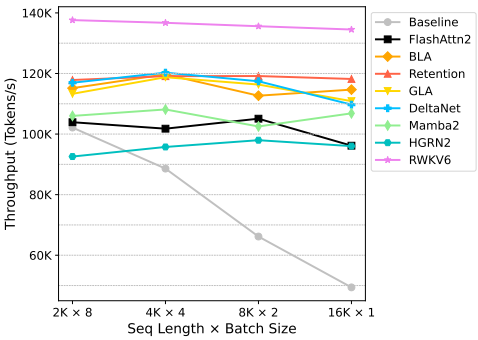

Figure 3: **Throughput (Tokens/s).** As sequence length increases, the throughput of Baseline declines significantly, whereas LSM models maintain stable training efficiency.

2.2.2 Hybrid Model Sequence Parallelism

Applying LASP-2 sequence parallelism to pure Linear-MoE models is straightforward, as this form of SP operates exclusively on the LSM modules, leaving the MoE layers unaffected. In hybrid Linear-MoE models, however, implementing SP becomes more complex due to the interleaving of distinct sequence modeling layers. To effectively optimize SP for these hybrid models, we introduce an integrated approach that incorporates SP across both the linear-MoE and standard transformer layers, thus enhancing overall efficiency. We illustrate the approach in Fig. 5 in Appendix A.2.1, and explain it as below:

On LSM Module. Similar to LASP-2, the SP for LSM modules is implemented via a single collective communication operation on the memory state $\mathbf{M}_s \in \mathcal{R}^{d \times d}$. This approach ensures that the communication complexity does not depend on either the sequence or sub-sequence length; rather, it scales only linearly with the SP size T, thereby maintaining efficiency in distributed setups.

On Standard Attention Module. Context parallelism (CP) is a SP technique used in Megatron-LM that divides input data and activations along the sequence dimension, specifically designed for standard softmax attention. Traditional CP implementations in Megatron-LM rely on a ring-like communication-computation overlap (Liu et al., 2023). In contrast, our approach for standard attention modules adopts the all-gather-based strategy used in the pretraining of Llama3 (Dubey et al., 2024). Rather than utilizing a ring strategy, we perform all-gather communication for $\mathbf{K}_s t$ and $\mathbf{V}_s$ tensors across devices, followed by local computation of attention output on each device's chunk of $\mathbf{Q}_s$. While all-gather communication theoretically has higher latency than ring-based methods, it offers greater flexibility and adaptability for handling different attention masks, such as document-level masks, making it ideal for varying attention patterns. Moreover, the latency of all-gather is minimized since the $\mathbf{K}_s$ and $\mathbf{V}_s$ tensors are notably smaller than the $\mathbf{Q}_s$ tensor, especially with grouped query attention (Ainslie et al., 2023). Consequently, the computational time for generating attention output significantly outweighs the cost of all-gather communication.

The Linear-MoE system also provides more training techniques like Tensor Parallelism, Hybrid Parallelism and Variable Length handling capacities, which are detailed in Appendix A.2. The concrete implementations of Linear-MoE system, including Modeling subsystem, Training subsystem and Evaluation module are described in Appendix A.3, due to space limitations.

3 Empirical Study

We perform experiments to evaluate the training efficiency of the Linear-MoE system, focusing on throughput and GPU memory requirements using eight A100 GPUs. For training the sparse MoE models, we set the EP size as 8. During the experiments, we maintain a total of 16K input tokens per iteration, while varying the input sequence lengths across {2K, 4K, 8K, 16K} with corresponding

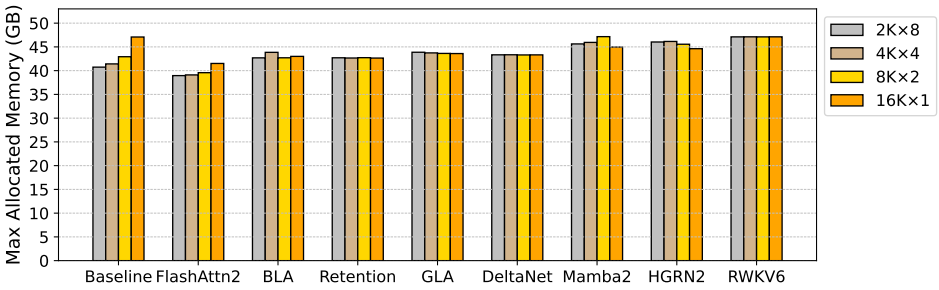

Figure 4: **Max Allocated GPU Memory (GB).** Legend presents *Seq length* $\times$ *Batch Size*. With the increasing of sequence length, the GPU memory requirements of Baseline and FlashAttention models grow quickly, while the LSM-based models keep almost consistent GPU memory consumption.

batch sizes of $\{8, 4, 2, 1\}$. As illustrated in Fig. 3 and Fig. 4, we observe that the standard attention Baseline shows a significant quadratic increase in memory usage and a decline in throughput as the input sequence lengths grow. FlashAttention-2 also demonstrates notable variations in both memory footprint and throughput, when the sequence length reaches 16K. In contrast, the Linear-MoE models, which incorporate LSM, exhibit relatively stable memory consumption and consistent throughput when sequence length increases, but number of input tokens remains fixed.

Furthermore, to highlight the efficiency benefits of the Linear-MoE training subsystem, we conduct ablation studies on MoE optimization techniques and parallelism training methods. The results of these experiments are presented in Table 4. It is evident that the implementation of MoE optimization techniques, specifically Grouped GEMM and MegaBlocks, significantly reduces the elapsed time for each iteration. Additionally, the various parallelism training techniques each demonstrate their own advantages in terms of memory footprint and overall training efficiency.

We provide detailed experiment setups and supplementary experiment results in Appendix A.4.

4 Conclusion

In this paper, we introduced Linear-MoE, a novel product-level system designed to integrate LSM with MoEs, aiming to advance both the efficiency and scalability of existing large models. By combining linear-complexity sequence modeling capabilities of LSM with sparsely activated MoE layers, Linear-MoE achieves high performance while addressing computational and memory constraints common in large model training and deployment. The dual subsystems: Modeling and Training, provide a flexible and extensible framework that supports diverse LSM methods and advanced parallelism techniques, including LASP-based sequence parallelism for handling long input sequences efficiently. Additionally, we explored hybrid models that further enhance adaptability by incorporating standard MoE Transformer layers. Our experimental results demonstrate that Linear-MoE achieves significant efficiency gains while maintaining strong performance across various benchmarks. These findings highlight the potential of Linear-MoE as the next generation of foundation model architecture.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*, 2024.

Soumith Chintala. Gpt-4 moe, June 2023. URL `https://x.com/soumithchintala/status/1671267150101721090`.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

OpenCompass Contributors. Opencompass: A universal evaluation platform for foundation models. `https://github.com/open-compass/opencompass`, 2023.

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

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

Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. Fewer truncations improve language modeling. *arXiv preprint arXiv:2404.10830*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

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

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

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022a.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022b.

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models, 2022.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.936. URL https://aclanthology.org/2023.findings-emnlp.936.

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

Hadi Pouransari, Chun-Liang Li, Jen-Hao Rick Chang, Pavan Kumar Anasosalu Vasu, Cem Koc, Vaishaal Shankar, and Oncel Tuzel. Dataset decomposition: Faster llm training with variable sequence length curriculum. *arXiv preprint arXiv:2405.13226*, 2024.

Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, and Yiran Zhong. Transnormerllm: A faster and better large language model with improved transnormer, 2024a.

Zhen Qin, Xuyang Shen, Weigao Sun, Dong Li, Stan Birchfield, Richard Hartley, and Yiran Zhong. Unlocking the secrets of linear complexity sequence model from a unified perspective. *arXiv preprint arXiv:2405.17383*, 2024b.

Zhen Qin, Weigao Sun, Dong Li, Xuyang Shen, Weixuan Sun, and Yiran Zhong. Lightning attention-2: A free lunch for handling unlimited sequence lengths in large language models. *arXiv preprint arXiv:2401.04658*, 2024c.

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

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

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Yikang Shen, Zhen Guo, Tianle Cai, and Zengyi Qin. Jetmoe: Reaching llama2 performance with 0.1 m dollars. *arXiv preprint arXiv:2404.07413*, 2024.

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

Weigao Sun, Disen Lan, Xiaoye Qu, and Yu Cheng. Lasp-2: Rethinking sequence parallelism for linear attention and its hybrid. 2024a.

Weigao Sun, Zhen Qin, Dong Li, Xuyang Shen, Yu Qiao, and Yiran Zhong. Linear attention sequence parallelism. *arXiv preprint arXiv:2404.02882*, 2024b.

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

Jamba Team, Barak Lenz, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, et al. Jamba-1.5: Hybrid transformer-mamba models at scale. *arXiv preprint arXiv:2408.12570*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang

Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

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

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

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

Jinle Zeng, Min Li, Zhihua Wu, Jiaqi Liu, Yuang Liu, Dianhai Yu, and Yanjun Ma. Boosting distributed training performance of the unpadded bert model. *arXiv preprint arXiv:2208.08124*, 2022.

Yujia Zhai, Chengquan Jiang, Leyuan Wang, Xiaoying Jia, Shang Zhang, Zizhong Chen, Xin Liu, and Yibo Zhu. Bytetransformer: A high-performance transformer boosted for variable-length inputs. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 344–355. IEEE, 2023.

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

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. *arXiv preprint arXiv:2406.16554*, 2024.

A APPENDIX

A.1 RELATED WORK

A.1.1 MIXTURE-OF-EXPERTS

MoE (Cai et al., 2024; Zhu et al., 2024) is gaining increasing attention in the development of large language models (LLMs) due to its ability to scale model size while maintaining computational efficiency. Its key strength lies in the sparse activation of experts and routing mechanisms, enabling a better balance between model performance and training cost. The effectiveness of MoE in modern deep learning was first demonstrated in Shazeer et al. (2017), where an MoE layer was introduced between LSTM layers, resulting in state-of-the-art performance on language modeling and machine translation benchmarks. Following this, the MoE layer was incorporated into the Transformer architecture, replacing the feed-forward network (FFN) layers. GShard (Lepikhin et al., 2020) applied MoE to Transformers, significantly improving machine translation across 100 languages. Switch Transformers (Fedus et al., 2022) further scaled model size to trillions of parameters, using a simplified and efficient MoE layer design. However, training MoE models often leads to load imbalance, where only a few experts are heavily utilized, leaving others underutilized. To address this, several strategies have been developed to optimize MoE training. These include the BASE layer (Lewis et al., 2021), the HASH layer (Roller et al., 2021), and Expert Choice (Zhou et al., 2022), all of which aim to maximize model capacity utilization. MoE architectures have been widely adopted in industry-leading models, such as Gemini-1.5 (Reid et al., 2024) and reportedly GPT-4 (Chintala, 2023). Other notable examples of LLMs incorporating MoE techniques include Mixtral (Jiang et al., 2024), DeepSeek V2 (Liu et al., 2024), Qwen2 (Yang et al., 2024a), JetMoE (Shen et al., 2024), Jamba (Team et al., 2024), and OLMoE (Muennighoff et al., 2024). Despite the advances in MoE, most research has focused on improving FFN layers and routers, while attention mechanisms have remained largely unchanged. There is still much room for exploring how to enhance the efficiency of MoE models by evolving their attention layers.

A.1.2 LINEAR SEQUENCE MODELING

Linear Attention. Linear attention encompasses a set of techniques aimed at calculating attention outputs using the "right-product kernel trick," which first computes key-value products, thereby avoiding the quadratic complexity associated with query-key computations. Vanilla linear attention (Katharopoulos et al., 2020) replaces the $\mathrm{Softmax}$ attention (Vaswani et al., 2017) with kernel methods, reducing the computational complexity to linear in relation to sequence length. Building on this, various extensions of linear attention have emerged. For example, TransNormerLLM (Qin et al., 2024a) introduces Lightning Attention, an optimized linear attention mechanism that speeds up processing by enhancing IO operations. Lightning Attention-2 (Qin et al., 2024c) further improves this by separately handling inter- and intra-block computations to fully exploit the advantages of linear attention on autoregressive tasks. RetNet (Sun et al., 2023) combines a retention mechanism with attention, offering both parallel training and linear-time inference. Gated Linear Attention (GLA) (Yang et al., 2023) introduces a data-independent gating mechanism and presents a hardware-efficient algorithm for training. DeltaNet (Schlag et al., 2021), along with its parallelized version (Yang et al., 2024b), applies a delta-rule-like update to improve performance in long-context scenarios. More recently, Gated Slot Attention (GSA) (Zhang et al., 2024), inspired by GLA, introduces a bounded-memory slot control mechanism within the gated linear attention framework, further boosting performance in tasks requiring strong recall abilities.

State Space Modeling. SSM provides a robust framework for capturing the behavior of sequence modeling within dynamic systems, and has demonstrated itself in the field of linear sequence modeling. Models such as S4 (Gu et al., 2022b) and its subsequent variants (Gu et al., 2022a; Gupta et al., 2022) have achieved notable success, particularly in long-range synthetic tasks. A recent example is Mamba (Gu & Dao, 2023), a representative SSM model that introduces a state selection mechanism. Mamba addresses the limitation of static dynamics in previous methods, arguing that they do not account for input-specific context selection within the hidden state, which is critical for tasks like language modeling. Mamba has shown superior performance compared to Transformers across various model sizes and scales. Mamba has been further refined in its successor, Mamba2 (Dao & Gu, 2024), which integrates a linear attention-like mechanism that improves hardware efficiency during training. Similar to how linear attention uses outer products to expand the state, Mamba2 leverages

a state-space duality that enables parallel attention-style computation while maintaining recurrent inference capabilities.

Linear RNN. Traditional RNNs struggle with long-context sequence modeling, largely due to their sequential nature during training, which limits their ability to benefit from scaling laws (Sun et al., 2023). To mitigate these issues, Linear RNNs introduce parallel training capabilities, achieving competitive performance with Transformers of comparable size. RWKV (Peng et al., 2023; 2024) is an example of a large language model based on linear RNNs, designed to effectively manage long-term dependencies. Furthermore, HGRN (Qin et al., 2024e) emphasizes the importance of data-dependent decay mechanisms in enhancing linear RNN performance, showing how tuning decay parameters can improve learning in long-context scenarios. The upgraded HGRN2 (Qin et al., 2024d) builds on this by introducing a state expansion mechanism that leverages outer product operations, allowing for better scalability and improved sequence modeling over extended inputs. Both RWKV and HGRN models aim to address the limitations of traditional RNNs for efficient long-sequence modeling.

A.2 More Training Techniques

A.2.1 Sequence Parallelism Algorithms

Algorithm 1 LASP-2 w/o Masking

Input: input sequence $\mathbf{X}$, distributed world size W, sequence parallel size $T = W$.
Distribute $\mathbf{X} = [\mathbf{X}_t]_1^T$.
for chunk $t \in \{1, \cdots, T\}$ on ranks $\{1, \cdots, W\}$ **in parallel do**
  Calculate $\mathbf{Q}_t = \mathbf{X}_t\mathbf{W}_Q$, $\mathbf{K}_t = \mathbf{X}_t\mathbf{W}_K$, $\mathbf{V}_t = \mathbf{X}_t\mathbf{W}_V$.
  Compute $\mathbf{M}_t = \mathbf{K}_t^\top\mathbf{V}_t$.
  Communicate $[\mathbf{M}_t]_1^T = \texttt{AllGather}([\mathbf{M}_t]_1^T)$.
  Compute $\mathbf{M}_{1:T} = \texttt{Sum}([\mathbf{M}_t]_1^T)$.
  Compute $\mathbf{O}_t = \mathbf{Q}_t\mathbf{M}_{1:T}$.
end for
return $\mathbf{O} = [\mathbf{O}_t]_1^T$.

Algorithm 2 LASP-2 w/ Masking

Input: input sequence $\mathbf{X}$, distributed world size W, sequence parallel size $T = W$.
Distribute $\mathbf{X} = [\mathbf{X}_t]_1^T$.
Initialize mask matrix $\mathbf{\Psi}$, where $\mathbf{\Psi}_{ij} = 1$ if $i \geq j$ and $\mathbf{\Psi}_{ij} = -\infty$ if $i < j$.
for chunk $t \in \{1, \cdots, T\}$ on ranks $\{1, \cdots, W\}$ **in parallel do**
  Calculate $\mathbf{Q}_t = \mathbf{X}_t\mathbf{W}_Q$, $\mathbf{K}_t = \mathbf{X}_t\mathbf{W}_K$, $\mathbf{V}_t = \mathbf{X}_t\mathbf{W}_V$.
  Compute $\mathbf{M}_t = (\mathbf{K}_t)^\top\mathbf{V}_t$.
  Communicate $[\mathbf{M}_t]_1^T = \texttt{AllGather}([\mathbf{M}_t]_1^T)$.
  Compute $\mathbf{O}_{\text{t,intra}} = [(\mathbf{Q}_t\mathbf{K}_t^\top) \odot \mathbf{\Psi}]\mathbf{V}_t$.
  Compute prefix sum
$$\mathbf{M}_{1:t-1} = \texttt{PrefixSum}([\mathbf{M}_t]_1^{t-1}).$$
  Compute $\mathbf{O}_{\text{t,inter}} = \mathbf{Q}_t\mathbf{M}_{1:t-1}$.
  Compute $\mathbf{O}_t = \mathbf{O}_{\text{t,intra}} + \mathbf{O}_{\text{t,inter}}$.
end for
return $\mathbf{O} = [\mathbf{O}_t]_1^T$.

A.2.2 Tensor Parallelism

The core computation mechanism of LSM modules can be abstracted in the following general form:

$$\mathbf{O} = \phi(\mathbf{Q})(\phi(\mathbf{K})^\top\mathbf{V}), \quad \mathbf{Q} = \mathbf{X}\mathbf{W}_q, \mathbf{K} = \mathbf{X}\mathbf{W}_k, \mathbf{V} = \mathbf{X}\mathbf{W}_v, \tag{5}$$

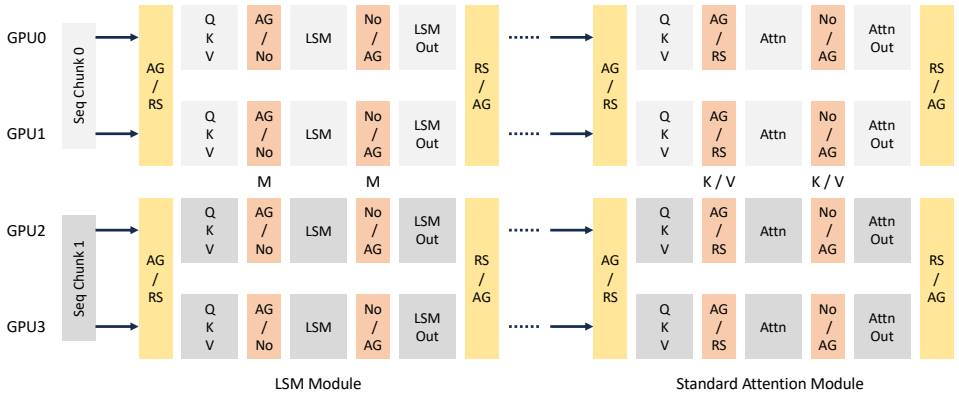

Figure 5: **Sequence Parallelism Approach on Hybrid Linear-MoE models.** We exemplify the parallelism on the hybrid layers of LSM and standard attention with both TP and SP (both have a dimension of 2). The communication operations colored in yellow and green are for TP and SP, respectively. AG/RS: all-gather in forward and reduce-scatter in backward, and vice versa. AG/No: all-gather in forward and no-op in backward, and vice versa. Note that the SP communication operations for linear attention operate on the single memory state $\mathbf{M}_s \in \mathcal{R}^{d\times d}$, while for standard attention, they operate on two states $\mathbf{K}_s, \mathbf{V}_s \in \mathcal{R}^{C\times d}$.

where TP is applied by splitting the matrix multiplications as follows:

$$\begin{aligned}\mathbf{Q} &= [\phi(\mathbf{X}\mathbf{W}_q^1), \phi(\mathbf{X}\mathbf{W}_q^2)], \mathbf{K} = [\phi(\mathbf{X}\mathbf{W}_k^1), \phi(\mathbf{X}\mathbf{W}_k^2)],\\ \mathbf{V} &= \mathbf{X}[\mathbf{W}_v^1, \mathbf{W}_v^2], \mathbf{O} = [\mathbf{O_1}, \mathbf{O_2}],\end{aligned} \tag{6}$$

where the weight matrices $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are divided along their columns, producing an output matrix $\mathbf{O}$ that is also split along columns.

The split output $[\mathbf{O}_1, \mathbf{O}_2]$ is then multiplied by an output linear weight matrix that is split along its rows, resulting in:

$$\mathbf{O} = [\mathbf{O}_1, \mathbf{O}_2][\mathbf{W}_o^1, \mathbf{W}_o^2]^\top \quad = \mathbf{O}_1\mathbf{W}_o^1 + \mathbf{O}_2\mathbf{W}_o^2, \tag{7}$$

which produces a unified output.

As with TP in standard attention, TP for LSM modules introduces an *all-reduce* collective communication operation during both the forward and backward passes. In practical terms, this all-reduce operation is implemented via two separate steps: all-gather and reduce-scatter, which together functionally achieve the same result as a single all-reduce.

A.2.3 Hybrid Parallelism

SP in Linear-MoE allows for a flexible choice of sequence parallel size that can be set to any factor smaller than or divisible by the total number of distributed nodes (i.e., the world size). This flexibility enables splitting input data across both batch and sequence dimensions, creating a combined approach known as data-sequence hybrid parallelism. Standard data parallelism techniques, such as Distributed Data Parallel (DDP) (Li et al., 2020), can integrate seamlessly with SP in Linear-MoE. Additionally, the sharded data parallelism method, like Distributed Optimizer (Korthikanti et al., 2022) in Megatron-Core, is also compatible.

Furthermore, the system provides support for TP, PP, and EP specifically tailored for Linear-MoE models. In the case of TP, its application to Linear-MoE models is direct and efficient, as detailed in §A.2.2. Regarding PP and EP, these parallelism techniques operate on Linear-MoE in much the same way as their original versions since they are not involved in the inner computations of the LSM modules but rather work at the level of complete Linear-MoE blocks or MoE layers. Moreover, TP, PP, and EP can be combined with DP and SP as introduced earlier, enhancing flexibility and scalability for large distributed setups.

A.2.4 Variable Length

During pretraining, batches generally consist of sequences with a uniform length. However, in the finetuning phase or during inference, the model often encounters batches containing sequences of

different lengths. A common approach to handle this variation is to right-pad each sequence in the batch so that all match the length of the longest sequence in that batch. While straightforward, this padding strategy can lead to inefficiencies, particularly when sequence lengths vary greatly within a batch. For standard transformers, more advanced methods have been introduced to address this issue. These methods include techniques like distributing workloads across GPUs to avoid padding altogether (Zeng et al., 2022; Zhai et al., 2023), or packing multiple sequences into a single batch while adjusting the attention mask as needed (Ding et al., 2024; Pouransari et al., 2024). In Linear-MoE, handling variable-length sequences is simplified by processing the entire batch as one continuous long sequence, effectively managing varying sequence lengths without padding.

A.3 Implementation

The implementation of the Linear-MoE system is based on Megatron-Core, an open-source library developed on PyTorch that incorporates optimized GPU techniques and advanced system-level enhancements. As depicted in Fig. 2, the Linear-MoE system consists of both modeling and training subsystems, facilitating adaptable model building and efficient training specifically for Linear-MoE models. Leveraging the capabilities of Megatron-Core, the Linear-MoE library is fully compatible with all NVIDIA Tensor Core GPUs, including support for FP8 acceleration on NVIDIA Hopper architectures.

The Linear-MoE design approach aims to minimize any invasive changes to Megatron-Core's source code. Rather than adding new modules directly, Linear-MoE operates independently, allowing users to benefit from the latest LLM practices without disruptions due to updates or changes within Megatron-Core.

A.3.1 Modeling Subsystem

Linear-MoE abstracts its LSM modules into modular and composable APIs, providing model developers and researchers with extensive flexibility to design and train large-scale custom Linear-MoE models on accelerated computing platforms. The system includes essential building blocks, such as core components for LSM mechanisms, MoE layers and Linear-MoE blocks, normalization techniques, and embedding methods. To enhance adaptability, LSM mechanisms are organized into three main categories: linear attention, SSM, and linear RNN, with multiple instances available in each. For linear attention, options include basic linear attention (BLA), Lightning Attention, Retention, GLA, DeltaNet, Based, and Rebased; for SSM, we provide Mamba2, the leading SSM model at present; and for linear RNN, options include HGRN2 and RWKV6. As LSM techniques evolve, Linear-MoE will continue to incorporate more LSM methods to ensure users have access to the latest advancements.

Additionally, Linear-MoE offers vital components such as a model library, tokenizers, model converters, usage examples, and a set of supportive toolkits. The model library includes instances of Linear-MoE models that are adapted from state-of-the-art open-source MoE architectures, including Qwen2 MoE, DeepSeekV2 MoE, and Mixtral MoE. These adapted instances are designated as Linear-MoE-Qwen2, Linear-MoE-DeepSeekV2, and Linear-MoE-Mixtral, respectively. These models are implemented following Megatron-Core format, with the standard attention layers replaced by LSM-based token mixing layers, while maintaining the original embedding, normalization, and expert layers unchanged.

A.3.2 Training Subsystem

Advanced parallelism techniques, encompassing tensor, sequence, pipeline, context, and MoE expert parallelism, are seamlessly incorporated into the Linear-MoE system through its design on top of the Megatron-Core library. This non-intrusive integration allows Linear-MoE to leverage the robust training capabilities of Megatron-Core, supporting large-scale model training across both standard attention layers and MoE expert layers. However, the inherent parallelism mechanisms, such as TP and SP, were not originally optimized for LSM modules. Additionally, Megatron-Core does not fully support efficient SP for hybrid models containing both LSM modules and standard attention layers. To address these gaps, we elaborate on our TP and SP approaches specifically designed for LSM modules and hybrid models, as discussed in §2.2.

Further capabilities, including mixed precision, activation recomputation, distributed optimizer, distributed checkpointing, and CPU offloading, are also inherited from Megatron-Core, enhancing model training flexibility and efficiency. And Linear-MoE supports 8-bit floating point (FP8) precision on Hopper GPUs, benefiting from the integration of NVIDIA's Transformer Engine (Micikevicius et al., 2022). This feature optimizes memory usage and accelerates performance during both training and inference stages.

To enhance the training speed of MoE layers, we incorporate MegaBlocks (Gale et al., 2023) into our Linear-MoE system. MegaBlocks is designed to optimize MoE training on GPUs by reconfiguring MoE computations using block-sparse operations and developing new block-sparse GPU kernels that effectively manage the inherent dynamism of MoEs. In addition, we also integrate the Grouped GEMM library into Linear-MoE, which introduces grouped GEMM kernels in PyTorch, thereby accelerating the computational processes involved in training MoE models.

A.3.3 Evaluation Module

In order to facilitate the evaluation on mainstream benchmarks, we have developed offline text generation of Linear-MoE models within the system. Based on this, mature evaluation frameworks such as OpenCompass (Contributors, 2023) and LM-Evaluation-Harness (Gao et al., 2023), are readily available for conducting evaluation tasks on Linear-MoE models. Furthermore, the system facilitates seamless bidirectional conversion between model weights from HuggingFace and Megatron-Core. This functionality enables users to easily leverage pretrained models from HuggingFace for continued pretraining or fine-tuning within the Megatron-Core environment. Additionally, it allows for the assessment of model performance by using HuggingFace's evaluation and inference pipelines on models trained within the Megatron-Core framework.

A.4 Supplementary Experiments

A.4.1 Experiment Setups

Models and Dataset. We conduct experiments on two Linear-MoE model series: A0.3B-2B and A1B-7B. A0.3B-2B denotes a Linear-MoE model containing a total of 2 billion parameters, with 0.3 billion parameters activated. The same applies for the A1B-7B model. Each series consists of several model instances, each incorporating a distinct instance of the LSM module. The specific LSM module instances used in our experiments include: basic linear attention (BLA) (Katharopoulos et al., 2020), Retentive Network (Retention) (Sun et al., 2023), Gated Linear Attention (GLA) (Yang et al., 2023), DeltaNet (Schlag et al., 2021), Mamba2 (Dao & Gu, 2024), HGRN2 (Qin et al., 2024d), and RWKV6 (Peng et al., 2023; 2024), all implemented in Triton. These model instances are evaluated against models with standard attention implementation in Megatron-Core (referred to as Baseline) and the FlashAttention-2 (Dao, 2023) implemented in Transformer Engine (in CUDA).

To implement the Linear-MoE model instances, we utilize the Qwen2 MoE architecture (Yang et al., 2024a) as the base model. All models are pretrained from scratch on a portion of the SlimPajama dataset (Soboleva et al., 2023). This dataset originally contains 627 billion tokens, we restrict our experiments to the first two chunks of the dataset, totaling approximately 100 billion tokens. The Qwen2 tokenizer is employed throughout the training processes.

Training Configurations. Table 2 details the training configurations for both Linear-MoE model series. We employ the Adam optimizer (Kingma & Ba, 2014) along with parallelism techniques, including TP and EP. Each pretraining run is performed on a node with eight A100 80G GPUs.

A.4.2 Training Loss Results

To evaluate the overall training performance of the Linear-MoE models, we pretrain the A0.3B-2B and A1B-7B model instances using 15B and 100B tokens, respectively. We test both pure and hybrid model configurations; for the hybrid models, we incorporate one quarter of standard transformer MoE layers throughout the architecture. For instance, in the 12-layer A0.3B-2B model, the hybrid configuration follows the pattern "LLLNLLLNLLLN", while the 16-layer A1B-7B model adopts the pattern "LLLNLLLNLLLNLLLN".

Models	A0.3B-2B	A1B-7B
Hidden Dimension	1024	2048
FFN Dimension	896	1024
Num of Heads	8	16
Num of Layers	12	16
Num of Act Experts	8	8
Num of Experts	64	64
LR	1e-4	1e-5
Minimum LR	1e-5	1e-6
LR Scheduler	Cosine	Cosine
Seq Length	2048	2048
Training Tokens	15B	100B

Table 2: **Linear-MoE Family Models and Training Configurations.** A0.3B-2B indicates that the Linear-MoE model has a total of 2 billion parameters, with 0.3 billion parameters activated. The same for A1B-7B.

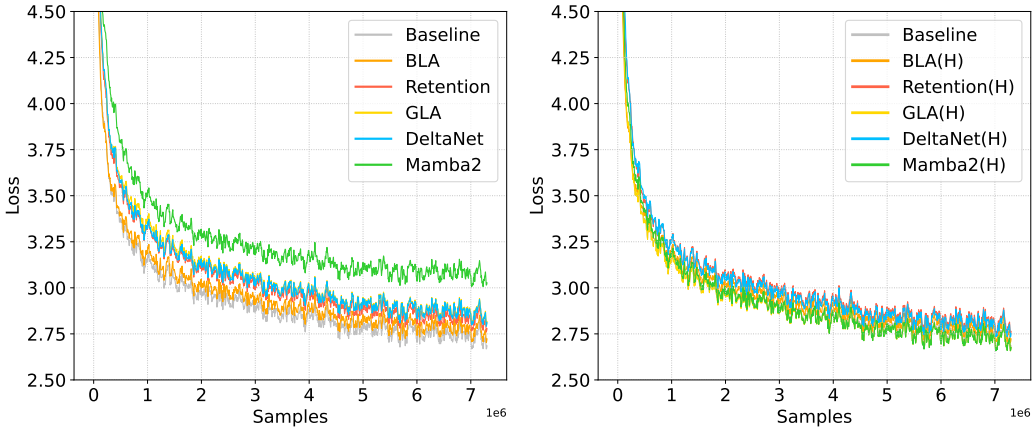

Figure 6: **Training Loss Curves of A0.3B-2B Model Instances.** Left: pure Linear-MoE models; Right: hybrid Linear-MoE models. Linear-MoE shows competitive training convergence performance compared to the standard attention Baseline.

The training loss curves for the A0.3B-2B model instances, which include both pure and hybrid Linear-MoE models, are presented in Fig. 6. The results demonstrate that the pure Linear-MoE architecture achieves competitive convergence performance compared to the standard attention Baseline. Moreover, the hybrid models exhibit more stable convergence and consistent performance when compared with the Baseline. Additional experiment results such as benchmark evaluations on A0.3B-2B models and training loss curves of A1B-7B models can be found in Appendix A.4.

A.4.3 Evaluation Results

To investigate the performance of the Linear-MoE system in the context of language modeling, we assess the accuracy on several benchmark datasets: PiQA (Bisk et al., 2019), HellaSwag (Hella.) (Zellers et al., 2019), WinoGrande (Wino.) (Sakaguchi et al., 2019), ARC-easy (ARC-e) and ARC-challenge (ARC-c) (Clark et al., 2018) under a zero-shot setting using OpenCompass (Contributors, 2023) framework. It is noteworthy that all models are trained from scratch without any data corruption.

The evaluation included both pure and hybrid versions of Linear-MoE A0.3B-2B model. Pure version contains exclusively LSM modules, while the hybrid version is stacked with a standard softmax attention layer follows three LSM layers for every four layers. As shown in Table 3, the trained Linear-MoE demonstrates competitive performance on language modeling tasks compared to the baseline (standard softmax attention). Both the pure and hybrid models, which integrate linear sequence modeling modules with the MoE architecture, achieve performance comparable to the baseline, demonstrating the effectiveness of the Linear-MoE system design.

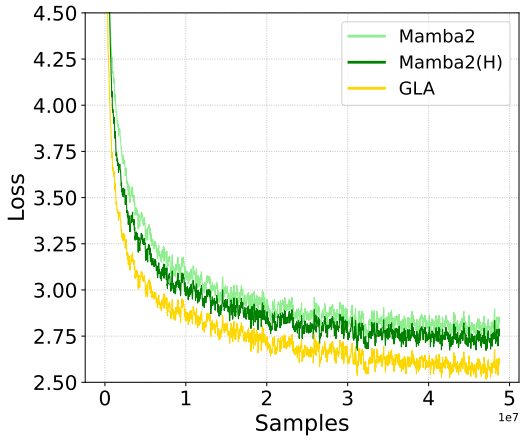

Figure 7: **Training Loss Curves of A1B-7B Model Instances.**

Model	**LSM Instance**	**PIQA**	**Hella.**	**Wino.**	**ARC-e**	**ARC-c**	**Avg.**
		acc ↑	acc_norm ↑	acc ↑	acc ↑	acc_norm ↑	↑
Baseline	Attention	41.78	23.89	44.36	23.81	24.41	31.65
Pure Linear-MoE	BLA	47.06	24.96	47.59	28.04	25.76	34.68
	Retention	14.80	8.39	14.52	8.64	7.80	
	GLA	6.26	3.37	6.16	3.70	4.07	4.71
	DeltaNet	16.16	9.26	16.42	10.23	8.81	
	Mamba2	44.07	23.35	43.17	25.22	23.05	31.77
	HGRN2	18.50	15.25	18.07	12.52	14.58	15.78
Hybrid Linear-MoE	BLA	18.39	8.52	29.52	11.46	6.78	14.93
	Retention	43.63	19.89	42.86	26.28	20.00	30.53
	DeltaNet	21.93	12.08	25.81	13.23	10.85	16.78
	Mamba2	41.35	23.58	40.73	21.16	24.41	30.25

Table 3: **Evaluation Results on Language Modeling Benchmarks (No Benchmark Data Corruption).** All models are pretrained from scratch on the same subset of the SlimPajama dataset with the Qwen2 tokenizer. No benchmark data corruption in the pretraining dataset. The A0.3B-2B hybrid Linear-MoE model has a stack as "LLLNLLLNLLLN", where "L" represents the Linear-MoE layer, and "N" represents the normal MoE transformer layer.

A.4.4 More Efficiency Results

MoE Optimization	Memory (GB)	Time/Iter (ms)
Baseline	35.28	1565.6
Grouped GEMM	35.01	455.4
MegaBlocks	36.90	348.8

EP	TP	PP	Memory (GB)	Time/Iter (ms)
1	1	1	35.28	1565.6
8	1	1	22.98	739.4
1	8	1	10.04	6879.0
1	1	8	8.89	1820.2
2	2	2	12.90	1684.9

Table 4: **Above: MoE Optimization. Below: Distributed training efficiency under different parallelism settings.** We report the memory usage per GPU (GB) and elapsed time per iteration (ms) while training the A0.3B-2B model with a sequence length of 2048 and a batch size of 4, using a node equipped with 8 A100 GPUs. The Baseline refers to the MoE implementation in Megatron-Core, which is used without any optimizations.

Seq Length $\times$ Batch Size	2K $\times$ 8		4K $\times$ 4		8K $\times$ 2		16K $\times$ 1	
	Mem.	Thpt.	Mem.	Thpt.	Mem.	Thpt.	Mem.	Thpt.
Baseline	40.74	102.14	41.42	88.60	42.93	66.17	47.08	49.39
Flash Attn	38.96	103.91	39.10	101.78	39.57	105.08	41.51	96.16
Basic LA	42.69	115.16	43.85	119.72	42.71	112.66	43.00	114.67
Retention	42.71	117.85	42.66	119.11	42.73	119.16	42.65	118.19
GLA	43.87	113.29	43.73	118.77	43.63	116.34	43.60	110.87
DeltaNet	43.33	116.95	43.34	120.27	43.31	117.43	43.32	109.72
Mamba2	45.63	105.99	45.94	108.13	47.16	102.51	44.97	106.84
HGRN2	46.03	92.58	46.14	95.74	45.56	97.98	44.97	96.02
RWKV6	47.11	137.62	47.12	136.73	47.11	135.60	47.12	134.51

Table 5: **Quantitative Efficiency Results.** We experiment on 8 A100 GPUs and report the max allocated GPU memory (GB) and throughput ($\times 10^3$ tokens/s) of A0.3B-2B model instances with varying input sequence lengthes and batch sizes.