Every Attention Matters: An Efficient Hybrid Architecture for Long-Context Reasoning

Ling Team

Bin Han, Caizhi Tang, Chen Liang, Donghao Zhang, Fan Yuan, Feng Zhu, Jie Gao, Jingyu Hu, Longfei Li, Meng Li, Mingyang Zhang, Peijie Jiang, Peng Jiao, Qian Zhao, Qingyuan Yang, Wenbo Shen, Xinxing Yang, Yalin Zhang, Yankun Ren, Yao Zhao, Yibo Cao, Yixuan Sun, Yue Zhang, Yuchen Fang, Zibin Lin, Zixuan Cheng, Jun Zhou

In this technical report, we present the Ring-linear model series, specifically including Ring-mini-linear-2.0 and Ring-flash-linear-2.0. Ring-mini-linear-2.0 comprises 16B parameters and 957M activations, while Ring-flash-linear-2.0 contains 104B parameters and 6.1B activations. Both models adopt a hybrid architecture that effectively integrates linear attention and softmax attention, significantly reducing I/O and computational overhead in long-context inference scenarios. Compared to a 32 billion parameter dense model, this series reduces inference cost to 1/10, and compared to the original Ring series, the cost is also reduced by over 50%. Furthermore, through systematic exploration of the ratio between different attention mechanisms in the hybrid architecture, we have identified the currently optimal model structure. Additionally, by leveraging our self-developed high-performance FP8 operator library–linghe, overall training efficiency has been improved by 50%. Benefiting from the high alignment between the training and inference engine operators, the models can undergo long-term, stable, and highly efficient optimization during the reinforcement learning phase, consistently maintaining SOTA performance across multiple challenging complex reasoning benchmarks.

Date: Oct 22, 2025

Hugging Face: https://huggingface.co/inclusionAI/Ring-flash-linear-2.0

Hugging Face: https://huggingface.co/inclusionAI/Ring-mini-linear-2.0

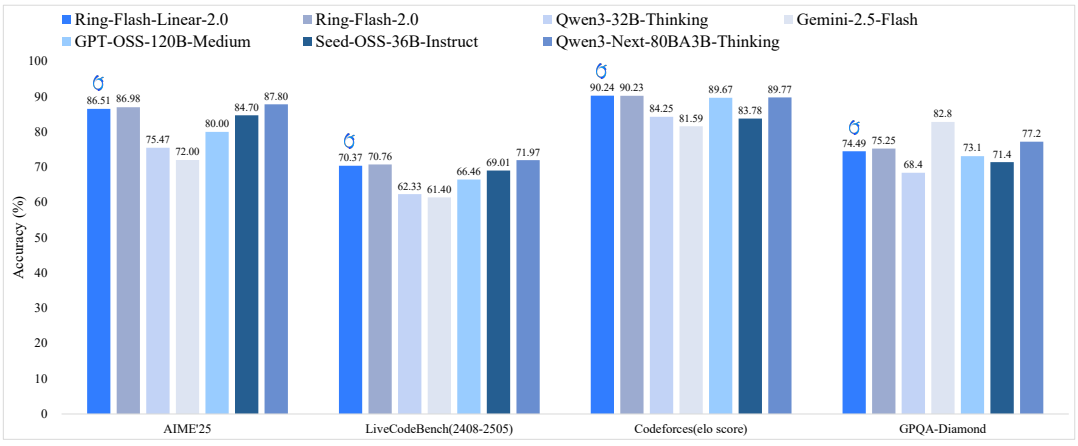


Figure 1 Benchmark results of Ring-flash-linear-2.0 versus counterparts on representative metrics.

1 Introduction

In recent years, Test-Time Scaling (Snell et al., 2024; Muennighoff et al., 2025) has significantly advanced the capability boundaries of Large Language Models(LLM). Models such as OpenAI's O-series [1], DeepSeek R1 (Guo et al., 2025), Gemini 2.5 (Comanici et al., 2025), and the Qwen (Yang et al., 2025) Thinking series have achieved enhanced performance by scaling the number of decode tokens. Breakthroughs in reasoning models have also contributed to improvements in non-reasoning tasks. At the same time, growing demands for long-context support in core applications such as Agent systems and code generation have made extended context capability a critical requirement for real-world model deployment.

Amid this progress, two major challenges have emerged: both the training length and output length of language models must be substantially increased, leading to a significant rise in overall resource consumption. Among the contributing factors, the attention mechanism stands out as a key bottleneck. Traditional architectures such as MHA (Vaswani et al., 2017), GQA (Ainslie et al., 2023), MQA (Shazeer et al., 2017) and MLA (Liu et al., 2024a) exhibit rapidly growing resource overhead as sequence length increases, imposing considerable pressure on both I/O and computation. Specifically, the computational complexity of traditional Softmax Attention grows quadratically with sequence length, while I/O overhead increases linearly with output length. These constraints severely hinder the ability of models to scale to longer contexts.

To address these challenges, research on Linear Attention has progressed rapidly. Approaches such as Mamba (Gu and Dao, 2024), Gated Linear Attention (Yang et al., 2023), and DeltaNet (Yang et al., 2024) have been introduced, achieving notable improvements in I/O and computational efficiency: computational complexity is reduced from $O(n^2d)$ to $O(nd^2)$, where n is the sequence length, d is the attention head dimension, and the space complexity of state memory becomes constant. These advances substantially enhance the efficiency of long-text processing and help control inference costs.

However, Linear Attention still exhibits certain limitations in practical applications:

- Pure linear language models often underperform in industrial-scale scenarios, particularly as model parameter counts and sequence lengths increase. To mitigate this, hybrid architectures have gained adoption as a balanced approach—retaining a portion of Softmax Attention in the model to maintain both efficiency and expressive power. Models such as Minimax M1 (Chen et al., 2025), GPT-OSS (Agarwal et al., 2025), and Qwen3-Next [2] have demonstrated promising results using this design.
- Although pure Linear Attention offers lower theoretical computational cost, its advantages only become pronounced at sequence lengths beyond 8K. However, the mainstream context length during pre-training typically remains in the 4K–8K range. Moreover, with the growing adoption of Mixture-of-Experts (MoE) architectures—which account for a high proportion of computations at lengths below 8K—the efficiency gains from Linear Attention during pre-training are further constrained.

To address these issues, we developed the Ring-linear model series based on a hybrid architecture, include two models: Ring-mini-linear-2.0 [3] and Ring-flash-linear-2.0 [4]. Our contributions are

[1] https://openai.com/index/introducing-o3-and-o4-mini
[2] https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Thinking
[3] https://huggingface.co/inclusionAI/Ring-mini-linear-2.0
[4] https://huggingface.co/inclusionAI/Ring-flash-linear-2.0

summarized as follows:

- We conducted a systematic study of hybrid linear architectures during pre-training and established a set of effective configuration guidelines.
- We implemented infrastructure-level optimizations for both pre-training and inference of hybrid linear models, achieving a 50% improvement in training performance compare with the native blockwise FP8 mixed-precision training method provided by Megatron [5] and a 90% increase in inference efficiency.
- Through systematic training-inference alignment for reinforcement learning, we significantly enhanced the stability of the model training process, and achieved stable long-horizon RL training.

In the remainder of this paper, we first present our model architecture in Section 2. Computational efficiency optimizations during training and inference are detailed in Section 3. We then describe the continued pre-training and post-training procedures in Sections 4 and 5, respectively. Finally, the performance of our models is evaluated in Section 6.

2 Model Architecture

In this section, we first introduce the basic architecture of Ring-linear. Subsequently, we discuss the specific design of the hybrid linear attention mechanism employed, along with several key architecture design choices. To achieve optimal performance within resource constraints and ensure economical training and efficient inference on longer sequences, we conducted extensive scaling law experiments and ablation studies to determine the final design. The key innovation of Ring-linear lies in adopting a highly sparse MoE architecture and utilizing linear attention to the greatest extent possible, as opposed to the traditional softmax attention typically used in standard transformers.

Size	Total Params	Active Params	Non-embedding Active Params	n_{layers}	d_{model}	$n_{experts}$	n_{top_k}	n_{heads}	n_{kv_heads}	Hybrid Ratio	Context Length
Mini	16.4B	1.6B	957M	20	2048	256	8	16	4	1:4	128K
Flash	104.2B	7.4B	6.1B	32	4096	256	8	32	4	1:7	128K

Table 1 Detailed specs of Ring-linear LLM family of models.

2.1 Basic Architecture

As shown in Figure 2, Ring-linear is composed of N layer groups, with each layer group consisting of M linear attention blocks and a Grouped Query Attention (GQA) block. Guided by the principles of Ling Scaling Laws (Tian et al., 2025a), Ring-linear incorporates a 1/32 activation-ratio MoE architecture, which has been systematically optimized across a range of design choices. These include expert granularity, the shared-expert ratio, attention balance, an auxiliary-loss-free approach (Wang et al., 2024) combined with a sigmoid routing strategy, Multi-Token Prediction (MTP) layers, QK-Norm, Partial-RoPE, and other advanced techniques. These innovations enable small-activation MoE models to deliver efficiency improvements of up to seven times compared to their dense counterparts. Specifically, the MLP layer in the first block of the model utilizes a dense MLP instead of a MoE-based MLP.

[5] https://github.com/NVIDIA/Megatron-LM

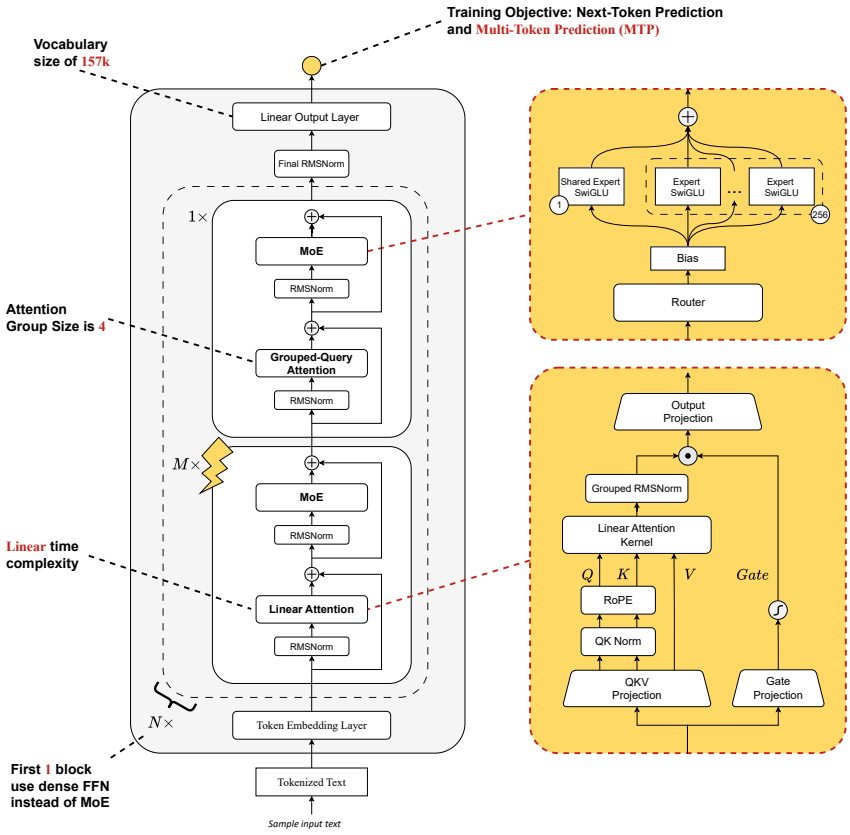


Figure 2 Model architecture of Ring-linear

Currently, two models based on the Ring-linear architecture have been open-sourced. These are the Ring-mini-linear-2.0, with a total of 16 billion parameters and 1.6 billion activated parameters (957M without embedding parameters), and the Ring-flash-linear-2.0, with a total of 104 billion parameters and 7.4 billion activated parameters (6.1B without embedding parameters). The detailed architectural settings of these two models are presented in Table 1.

2.2 Hybrid Linear Attention

Although softmax attention has been widely adopted in various LLMs, it suffers from quadratic computational complexity with respect to sequence length and requires Key-Value (KV) cache storage that scales linearly with sequence length. These limitations hinder both training and inference efficiency, especially in long-text scenarios such as test-time scaling and long-context retrieval tasks.

2.2.1 Linear Attention

We found that linear attention, which requires constant KV cache storage and incurs computational cost that scales linearly with sequence length, can serve as a viable and significantly more efficient alternative to softmax attention. In Ring-linear, we employ Lightning Attention (Qin et al., 2023) as

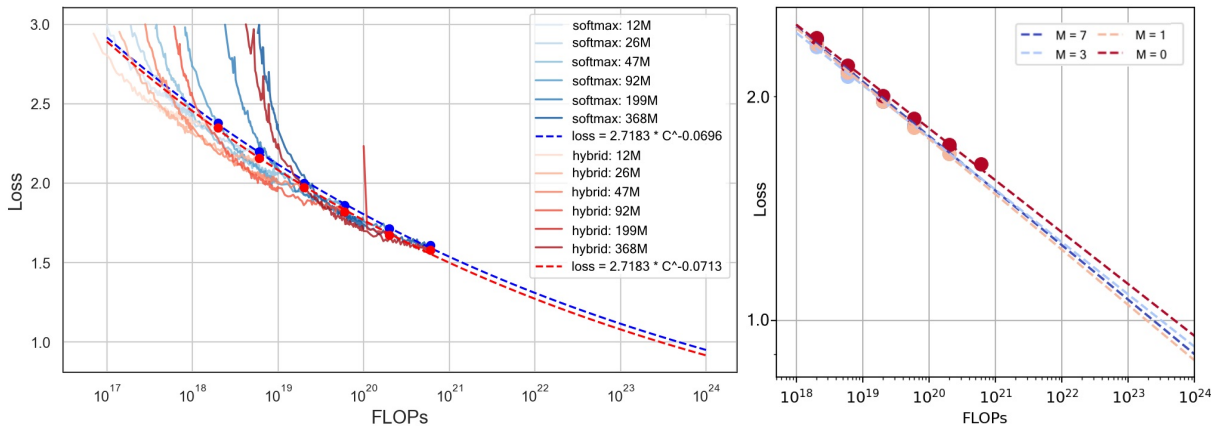


Figure 3 Scaling law curves. (Left) Curves fitted with different runs of the hybrid linear architecture ($M = 1$) and the softmax attention architecture. (Right) Curves comparing different layer group size (M) configurations.

the specific implementation of linear attention. Its mechanism can be formulated as follows:

$$\mathbf{O} = \mathbf{Q}(\mathbf{K}^{\mathrm{T}}\mathbf{V}), \tag{1}$$

where $\mathbf{O}$, $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V} \in \mathbb{R}^{n \times d}$ are the output, query, key, and value matrices, respectively, with n denoting sequence length and d representing feature dimension. This linear formulation facilitates recurrent prediction with a commendable complexity of $\mathcal{O}(nd^2)$.

During the forward pass of Lightning Attention, the output of the t-th token can be formulated as

$$\mathbf{o}_t = \mathbf{q}_t \sum_{s \le t} \lambda^{t-s} \mathbf{k}_s^{\mathrm{T}} \mathrm{v}_s. \tag{2}$$

In a recursive form, the above equation can be rewritten as

$$\begin{aligned} \mathbf{kv}_0 &= 0 \in \mathbb{R}^{d \times d}, \\ \mathbf{kv}_t &= \lambda \mathbf{kv}_{t-1} + \mathbf{k}_t^{\mathrm{T}} \mathbf{v}_t, \\ \mathbf{o}_t &= \mathbf{q}_t(\mathbf{kv}_t), \end{aligned} \tag{3}$$

where

$$\mathbf{kv}_t = \sum_{s \le t} \lambda^{t-s} \mathbf{k}_s^{\mathrm{T}} \mathrm{v}_s. \tag{4}$$

The matrix $\mathbf{kv}_t \in \mathbb{R}^{d \times d}$ defined in Equation 4 serves as the KV cache in Lightning Attention. In contrast to the linear growth of the KV cache in softmax attention, it requires only constant storage throughout the whole generation process.

2.2.2 Hybrid Architecture

Despite achieving performances comparable to that of softmax attention on most downstream tasks, linear attention underperforms in retrieval. In contrast, the hybrid linear attention architecture not only matches but also surpasses the retrieval and extrapolation capabilities of the pure softmax attention architecture (Li et al., 2025).

We evenly divide the model layers into several groups, each containing layer group size $(M + 1)$ layers. In each layer group, one softmax attention layer follows M linear attention layers. When $M = 0$, the architecture reduces to a pure softmax attention architecture.

To compare the hybrid linear attention architecture with the softmax attention architecture, we fit scaling law curves for various layer group size configurations. Following Chinchilla's methodology (Hoffmann et al., 2022), we establish power-law relationships between training FLOP budget and the training loss. The resulting scaling law curves are shown in Figure 3. As observed, the hybrid linear architecture consistently outperforms the pure softmax attention architecture. Moreover, a large layer group size (e.g., $M = 7$) performs well under high FLOP budgets. Therefore, to strike a balance between efficiency and effectiveness, we set layer group size to 8 ($M = 7$) and 5 ($M = 4$) for Ring-flash-linear-2.0 and Ring-mini-linear-2.0, respectively.

2.2.3 Key Architecture Design Choices

As mentioned earlier, we conducted extensive ablation experiments on the specific implementation details within the linear attention module to determine the optimal architectural design. Our design principle focuses on achieving the best model performance while ensuring efficient distributed training and decoding. Below are some of the key architectural design choices we explored.

Grouped RMSNorm To avoid the all-reduce operations required by a standard RMSNorm layer between the linear attention kernel and output projection under tensor parallelism (size > 1), we employ a grouped normalization strategy. This approach performs RMSNorm locally within each rank, eliminating communication in both the forward and backward passes.

Rotary Position Embedding (RoPE) As shown on the right side of Figure 2, after performing the normalization operation on the Q and K inputs within the linear attention module, we applied the RoPE operation (which was applied to only half of the dimensions). Our experiments revealed that incorporating RoPE resulted in a reduction of approximately 0.004 in the training language model (LM) loss.

Head-wise Decay Our experiments found that the performance of the hybrid linear model is particularly sensitive to the decay coefficient of the hidden state in the Linear Attention mechanism. In the Lightning Attention setting, using a power-law decay rate for the head-wise decay, as opposed to a linear decay rate, resulted in a reduction of approximately 0.04 in the training LM loss. This improvement also had a significant impact on the performance of downstream tasks.

2.2.4 Cost Analysis for Decoding

In practical LLM applications and reinforcement learning (RL) training, decoding efficiency often serves as a critical bottleneck for model test-time scaling. During the LLM decoding process, the limited memory bandwidth of accelerators such as GPUs makes the size of the attention mechanism's KV cache access a key factor affecting decoding speed. Existing approaches, such as GQA (Ainslie et al., 2023) and MLA (Liu et al., 2024a), similarly aim to improve decoding efficiency by optimizing KV cache memory access. In Figure 4, we present how the KV cache/State memory access size of our proposed architecture scales with sequence length, along with a comparison against other related methods.

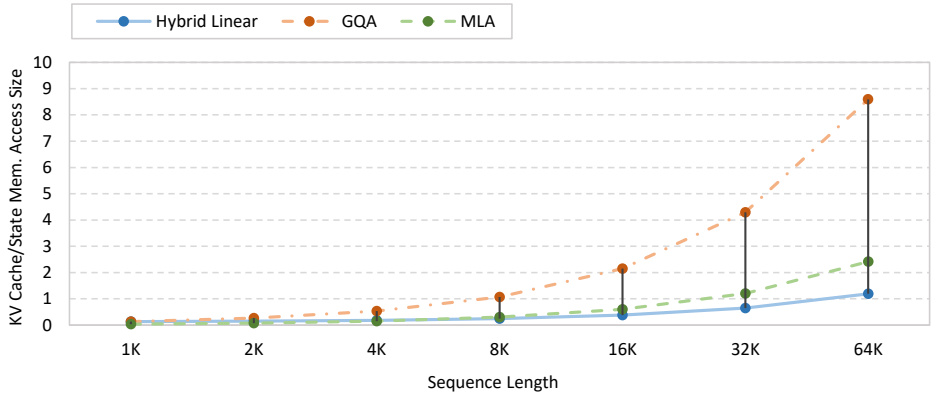


Figure 4 Variation of KV cache/State memory access size with increasing sequence length.

3 Computation Optimization

To fully demonstrate the computational efficiency advantages of the hybrid linear model, we conducted comprehensive performance optimizations from both training and inference perspectives. The core component of these optimizations lies in kernel fusion and optimization. Additionally, for training, we further implemented kernel fusion and introduced more fine-grained recomputation strategies tailored for the FP8 mixed precision training, significantly improving the training efficiency. On the inference side, we introduced speculative decoding tailored for hybrid linear models, effectively reducing the response latency under low concurrency scenarios. The overall optimization architecture is illustrated in Figure 5.

3.1 GPU Kernel Optimization

We performed extensive kernel fusion and optimization for both training and inference. Kernel fusion not only reduces computational latency but also decreases activation memory consumption during training. Lower memory usage enables larger micro-batch sizes. In our experiments, we observed that increasing the micro-batch size from 1 to 2 or from 2 to 4 in MoE training resulted in over 20% improvement in training efficiency. We briefly describe several key kernels below; for full implementation details, please refer to our open-source code repository[6] [7].

Linear Gate: In linear attention layers, we fused the operations related to the gating mechanism (e.g., attention output transpose, group RMS norm, gate sigmoid, and multiplication) into a single kernel. This reduces multiple memory accesses to the GPU memory and lowers activation memory consumption during training.

Permute/Unpermute: Compared to the approach in Megatron, which involves padding/unpadding and permute/unpermute separately, we adopted a more efficient strategy by modifying the routing map directly. This allows us to integrate padding/unpadding into the permute/unpermute operations. Moreover, in extreme cases (e.g., when the number of tokens assigned to a GPU is less than the padding size), this approach does not incur additional overhead.

QK Norm + Partial RoPE: The Ring models incorporate operations such as QK normalization and partial RoPE. We fused the QK normalization with the preceding split, RoPE, and transpose operations.

[6] linghe: https://github.com/inclusionAI/linghe
[7] flood: https://github.com/alipay/PainlessInferenceAcceleration/tree/main/flood

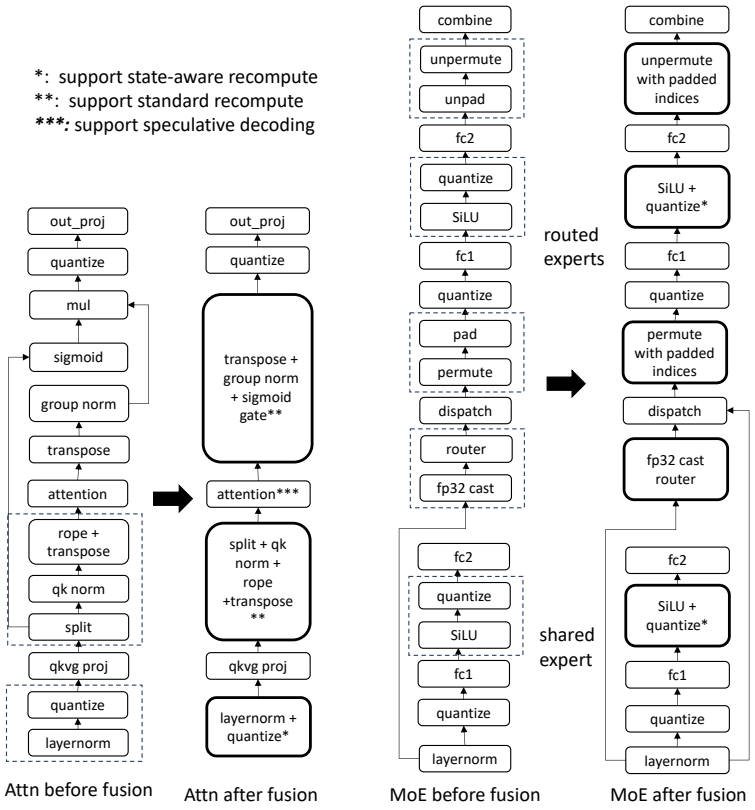


Figure 5 Overview of computation optimization

MoE Router: The baseline approach requires casting hidden states to FP32 before computing the router, which significantly increases both I/O and activation memory. We instead perform cast and compute within the fused kernel, outputting results in FP32 (forward) or BF16 (backward).

Linear Attention: Existing linear attention implementations (e.g., fla.chunk_simple_gla_fwd [8]) use 2–4 kernels for the prefill stage, leading to high kernel launch overhead and memory traffic. To address this, we redesigned the partitioning strategy using partitioned Q/K and V, improving performance without sacrificing parallelism. This allows us to use only one Triton kernel (and optionally one additional kernel if Q/K are split).

Other Kernel Fusions: We further optimized kernels based on the model size and parallelization strategy of the Ring model. For example, since we did not use tensor parallelism during training, we were able to develop a more efficient cross-entropy loss kernel without considering distributed loss computation. We also optimized batch computation for operations such as counting zero and calculating gradient norm, achieving significant performance improvements.

3.2 FP8 training optimization

Compared to BF16 GEMM, FP8 GEMM offers significantly improved computational speed. However, converting BF16 inputs to FP8 introduces quantization overhead, which becomes a bottleneck, especially on high-performance GPUs like the H800, where quantization can account for nearly 20% of the GEMM time. Therefore, fusing quantization with adjacent kernels becomes a key strategy to reduce overall latency.

[8]https://github.com/fla-org/flash-linear-attention

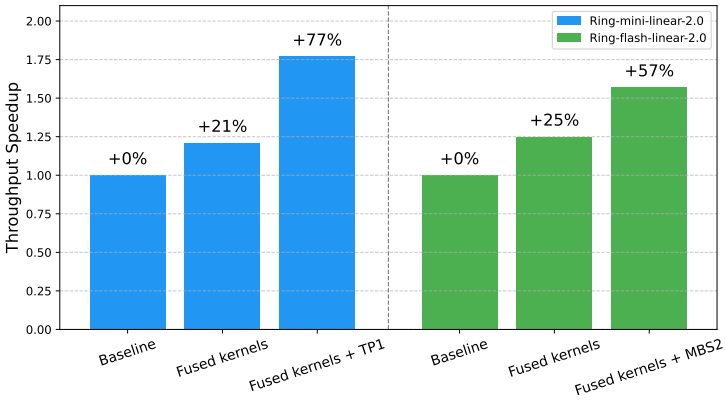


Figure 6 Speedup of LingHe and optimized parameters

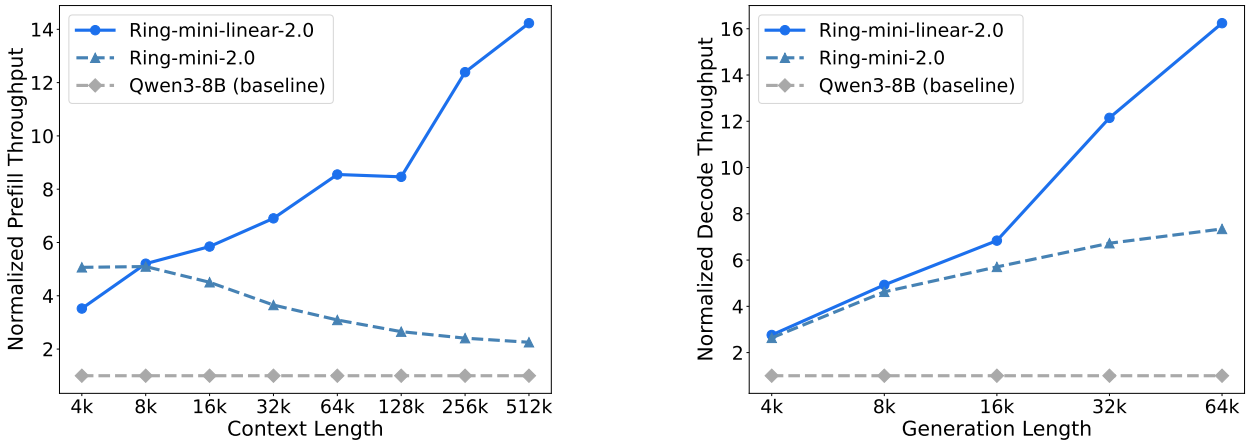
Quantization Fusion: Quantizaiton layers are usually placed after activation or normalization layers. For example, the SiLU is immediately followed by a Linear layer. We modified the SiLU kernel to directly output the quantized tensor, eliminating the need to write and read the BF16 output of SiLU. Assuming the input shape is [M, N], the I/O volume of the non-fused version is approximately $8MN$, while the fused version reduces it to $4MN$. This leads to significant acceleration for I/O-bound kernels. Similarly, in the backward pass, we optimized operations such as SiLU backward to directly output quantized dy and dy^T, reducing kernel time by nearly half.

State-aware recompute: We further optimized fine-grained recomputation. In the forward pass, the quantized x is needed to compute $y = xw$. However, in the backward pass, the transposed quantized x^T is required to compute $dw = x^T y$. Therefore, the forward pass in recomputation can have different computation and quantization logic compared to the regular forward pass. Leveraging this insight, we designed the forward pass to compute and output only the quantized x when not in recomputation, and only the quantized x^T when in recomputation, thereby reducing redundant computation.

3.3 Training efficiency

The training efficiency with our optimized kernels and parameters was evaluated, as shown in Figure 6. For Ring-mini-linear-2.0, the baseline configuration for FP8 blockwise implementation used a global batch size (GBS) of 4416, micro-batch size (MBS) of 4, tensor parallelism (TP) of 2, and 32 H800 GPU. Our optimized version with fused kernels used GBS=4352, MBS=4, and TP=1 (the slightly smaller GBS is due to the inability to reach 4416 with 32 GPUs and MBS=4). Thanks to the memory-efficient fused kernels and more aggressive recomputation, we were able to support MBS=4 with TP=1, resulting in a 77% improvement in training throughput compared to the baseline.

For Ring-flash-linear-2.0, the baseline configuration used GBS=8352, MBS=1, pipeline parallelism (PP)=6, virtual pipeline parallelism (VPP)=2, and 288 H800 GPUs. Our optimized version with fused kernels achieved a 25% improvement in training throughput over the baseline. By leveraging the saved memory, we increased the micro-batch size from 1 to 2 and adjusted the pipeline configuration from PP=6/VPP=2 to PP=8/VPP=1, ultimately achieving a 57% improvement in overall throughput.



(a) The normalized prefill throughput.

(b) The normalized decode throughput.

Figure 7 The inference efficiency of Ring-mini-linear-2.0 (TP=1, 1×H20).

3.4 Inference efficiency

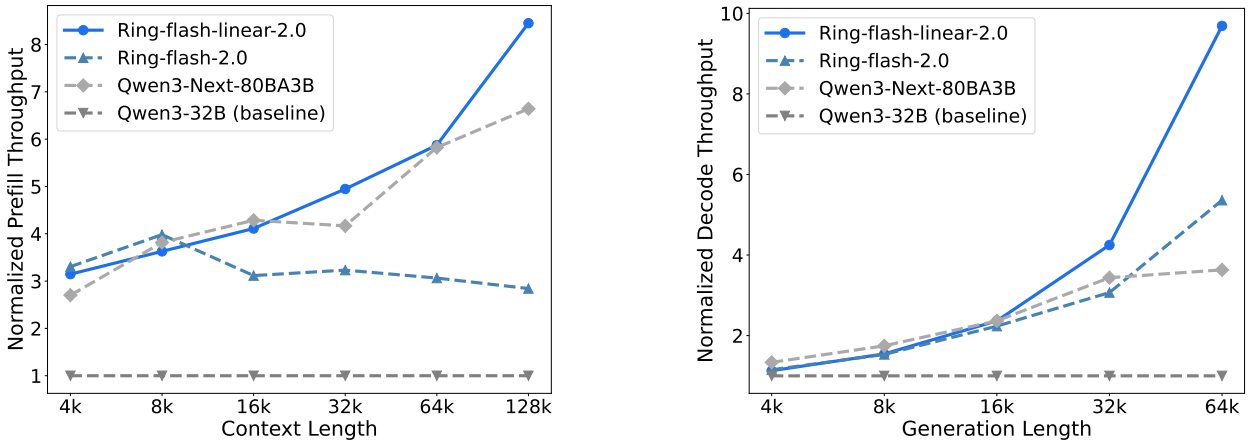
Although linear attention is theoretically more computationally efficient, its practical implementations have been limited by the absence of support for advanced inference solutions such as SGLang and the vLLM V1 framework. Moreover, the decoding kernel is often fragmented into multiple operations, which further diminishes overall efficiency. To overcome these limitations, we developed a set of optimized fused linear attention kernels and integrated them into both SGLang and vLLM for the Ring-linear models. These improvements not only extend support for a broader variety of inference modes but also substantially enhance system throughput.

Using the SGLang framework, we conduct a comprehensive evaluation of the inference efficiency of the Ring-linear models—without MTP—by measuring both prefill and decode throughput. Prefill throughput refers to the number of input tokens processed per second at a batch size of 1, whereas decode throughput denotes the number of output tokens generated per second at a batch size of 64. To facilitate comparison across models, we report normalized throughput relative to a baseline model. The inference performance of Ring-mini-linear-2.0 and Ring-flash-linear-2.0 is presented in Figure 7 and Figure 8, respectively. To comprehensively illustrate the inference efficiency of Ring-linear models in the figures, we compare them against dense model counterparts (Qwen3-8B and Qwen3-32B (Yang et al., 2025)) and their corresponding Ring-2.0 series (Ring-mini-2.0 [9] and Ring-flash-2.0 [10]) equipped with softmax attention. Specifically, for Ring-flash-linear-2.0, we conduct an in-depth comparison with Qwen3-Next-80BA3B, which also employs the hybrid linear attention architecture.

During the prefill phase, Ring-linear begins to outperform Ring-2.0 once the context length exceeds 8K, after which its throughput scales rapidly. Ultimately, it achieves more than 2.5 times the throughput of Ring-2.0 and over 8 times that of the baseline models for context lengths beyond 128K. In the decode phase, the Ring-linear models maintain strong performance, surpassing Ring-2.0 when generation length exceeds 4K. At 64K context length, they deliver more than twice the throughput of Ring-2.0 and exceed baseline performance by over tenfold.

Additionally, large language models can benefit from speculative decoding to accelerate infer-

[9] https://huggingface.co/inclusionAI/Ring-mini-2.0
[10] https://huggingface.co/inclusionAI/Ring-flash-2.0



(a) The normalized prefill throughput.

(b) The normalized decode throughput.

Figure 8 The inference efficiency of Ring-flash-linear-2.0 (TP=4, 4×H20).

ence for small-batch requests. However, existing linear attention kernels in the community do not support custom attention masks, making tree-based speculative decoding infeasible. We developed the first linear attention kernel that supports tree masks and collaborated with our tree-mask-compatible attention kernel to enable speculative decoding for the Ring hybrid linear models. This functionality is already available in our offline inference framework Flood, and we are currently working on porting the kernel to SGLang and integrating it with our previous work lookahead (Zhao et al., 2024).

4 Continued Pre-Training

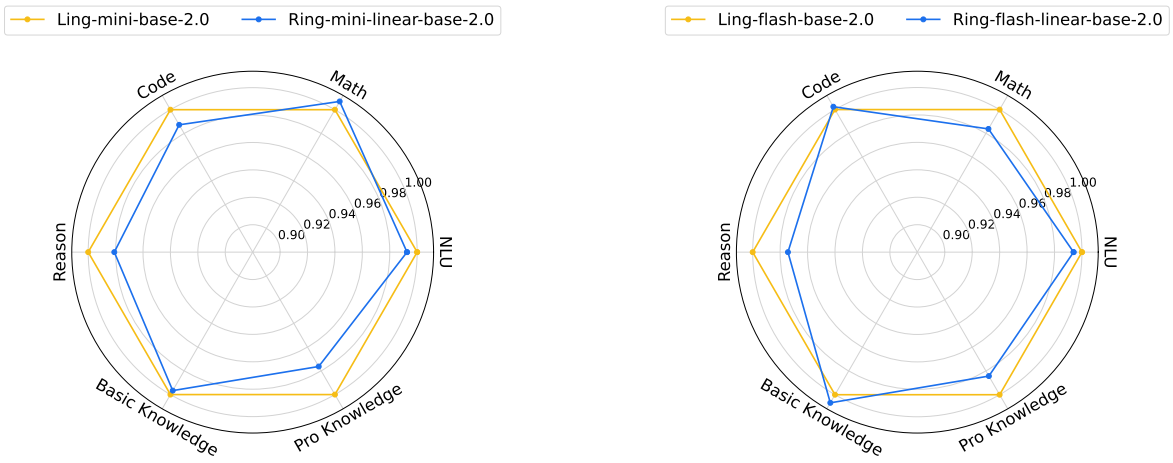
To reduce training costs, the base models of the Ring-mini-linear-2.0 and Ring-flash-linear-2.0 models are initialized from Ling-mini-base-2.0-20T [11] and Ling-flash-base-2.0-20T [12], respectively. Specifically, the QKV projection in each Linear Attention layer is converted to MHA by expanding parameters along the head dimension, while additional parameters introduced by the gate projection and RMSNorm are randomly initialized. Starting from the above two initialized models, a two-stage continued pre-training strategy is employed to restore the capabilities of the original base models.

Continued training Stage This stage samples data from the same corpus used for the Ling-base-2.0-20T models and is trained with 4K context length to restore the base model's capabilities. During this stage, Ring-mini-linear-base-2.0 is trained on 600B tokens, while Ring-flash-linear-base-2.0 is trained on 1T tokens.

Mid-training Stage It follows the same strategy as Ling-mini-base-2.0 and Ling-flash-base-2.0. To prepare for the subsequent post-train process, we progressively extend the context window of the models from 4K to 32K and finally to 128K, while correspondingly increasing the proportion of high-quality reasoning data.

During the above training process, we reuse the critical hyperparameters from Ling-base-2.0 and substitute the standard Warmup-Stable-Decay (WSD) learning rate scheduler (Hu et al., 2024b)

[11] https://huggingface.co/inclusionAI/Ling-mini-base-2.0-20T
[12] https://huggingface.co/inclusionAI/Ling-flash-base-2.0



(a) Ring-mini-linear-base-2.0

(b) Ring-flash-linear-base-2.0

Figure 9 The normalized performance after continued pre-training.

with the Warmup-Stable-Merge (WSM) scheduler (Tian et al., 2025b). By merging mid-training checkpoints to simulate learning rate decay strategy, the resulting Ring-linear-base-2.0 models perform on par with Ling-base-2.0 models trained on 20T tokens across various benchmarks. Figure 9 illustrates the performance of Ring-linear-base-2.0 on different capabilities, including coding, mathematics, reasoning, knowledge, and Natural Language Understanding (NLU). The normalized performance refers to the normalized scores of Ring-linear-base-2.0 relative to Ling-base-2.0. As demonstrated in the figure, Ring-linear-base-2.0 models restore more than 98% of the original models' performance in most categories. However, the models exhibit minor deficiencies in reasoning and professional knowledge tasks, which may be attributed to the knowledge forgetting problem (Ibrahim et al., 2024) during the continued pre-training process.

5 Post-Training

Following the continued pre-training process, we further enhance the model's capabilities through a post-training phase, which primarily consists of two stages: Supervised Fine-Tuning (SFT) and RL. It is noteworthy that we have identified the root cause of the prevalent problem of training collapse in RL: *training-inference disparity*. By achieving systematic alignment between training and inference, we have realized long-term stable RL training.

5.1 Supervised Fine-Tuning

The SFT data of Ring-linear places greater emphasis on comprehensive and balanced reasoning capabilities and generalization ability. In addition to including high-difficulty SFT data in areas such as mathematics, coding, science, and logic, the dataset also incorporates general-purpose SFT data covering knowledge, agent tasks, subjective creation, and medical domains. Notably, we have re-synthesized the Function Calling SFT data to better align with more general Function Calling patterns. As a result, Ring-linear series, as reasoning-oriented models, not only perform exceptionally well on reasoning tasks but also provide satisfactory responses to non-reasoning tasks across multiple domains.

All SFT data has undergone stricter de-noising and de-toxification processes, including n-gram filtering and semantic similarity detection, to ensure model safety and the authenticity of its

capabilities. Following the previous stage, the SFT training process is with context window of 128K. To prevent overfitting during the SFT phase and to provide ample room for subsequent RL, we selected a checkpoint of an earlier epoch (not the one with the highest benchmark score) for the downstream RL stage.

5.2 Reinforcement Learning

Building upon SFT, we subsequently conducted RL across multiple domains including mathematics, coding, science, logic, and subjective tasks. All of the samples underwent meticulous screening, from which samples with an appropriate difficulty level were selected for RL training. In this phase, the model is trained with a sufficiently long context window (e.g., 64K) to strike an optimal balance between performance and efficiency. We observed that with high-difficulty training data, smaller windows (e.g., 32K) introduce potential limitations, namely a high truncation rate and a lower performance ceiling. Conversely, training with a sufficiently long context window proves to be a superior choice, yielding a negligible truncation rate and enabling the model to reach a higher performance ceiling.

Unlike pre-training and SFT, RL for large language models relies on both training and inference. We observe that even standard components in large language models, such as RMSNorm and RoPE, exhibit non-negligible implementation discrepancies across common training (e.g., Megatron, FSDP) and inference (e.g., vLLM, SGLang) frameworks. These discrepancies accumulate and amplify layer by layer, leading to significant differences between training and inference (rollout) outcomes. In extreme cases, the output probability for the same token can be 0 during training and 1 during inference. Such training-inference disparity undermines the theoretical assumption of on-policy, consequently causing instability in RL training and limiting its performance.

Moreover, two prevalent factors notably exacerbate this training-inference disparity:

- MoE Architecture: Discrepancies in the selection of experts between training and inference introduce large errors.
- Long-CoT Models: Longer model outputs lead to greater cumulative errors.

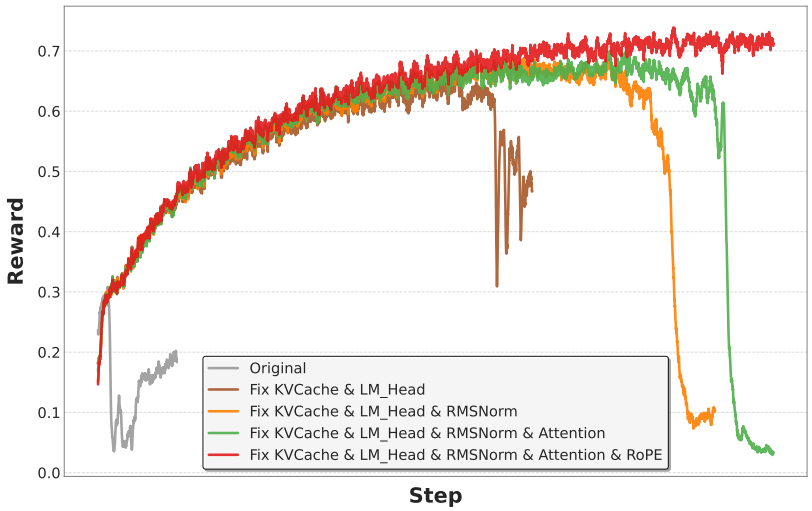


Figure 10 The ablation experimental results of the RL long-term training curves after correcting the training-inference disparity in different modules of the hybrid linear model.

Therefore, the challenges in RL, presented by long-output MoE models and short-output Dense models, are fundamentally different. It is frequently observed that algorithms effective for the latter may not necessarily apply to the former, and vice versa.

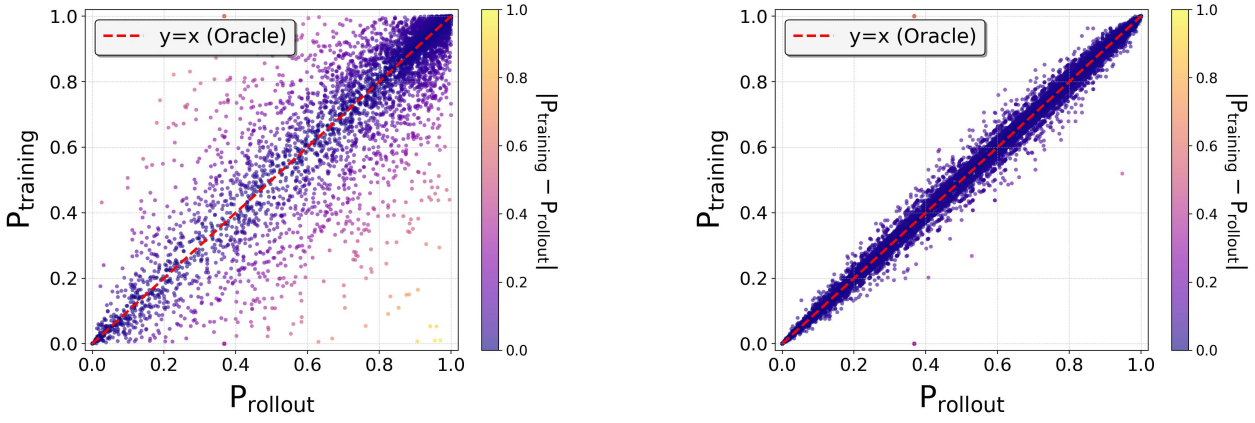
Although this issue has recently gained attention in some studies (Zheng et al., 2025; Yao et al., 2025), most efforts have focused on algorithmic mitigation. In our work, we not only improve RL stability from an algorithmic perspective but also devote substantial effort to systematically addressing the training-inference disparity, aiming to fundamentally solve the problem. As shown in Figure 10, each aligned module contributes to improved training efficiency and stability in RL.

5.2.1 Systematic Training-Inference Alignment

To align training and inference, the same input must be provided to both the training and inference frameworks, with activations compared module-by-module and layer-by-layer. Discrepancies in each activation value should be systematically identified and addressed from front to back. The overall alignment process can be divided into three stages:

- Alignment of prefill in training with prefill in inference.
- Alignment of prefill in training with decode in inference.
- Alignment under different parallelization configurations.

To ensure consistency between training and inference, our efforts generally focus on three aspects: ensure identical implementation, maintaining appropriate precision, and eliminating nondeterminism. Given the complexity of the full alignment workflow, the following paragraphs highlight specific points within critical modules that are particularly prone to training-inference disparity.



(a) Before **(b)** After

Figure 11 Token output probability distributions before and after KVCache correction.

KV Cache In hybrid linear models, the KV states in linear attention require accumulation (Equation 4), which demands higher numerical precision (e.g., FP32). If the KVCache is initialized as BF16 in the inference engine, errors will accumulate progressively during the recurrent process, leading to significant precision divergence, as shown in Figure 11.

LM Head The softmax function is highly sensitive to numerical precision, necessitating FP32 for the `lm_head` layer. However, directly configuring the `lm_head` to FP32 in the training engine can introduce substantial computational and memory overhead. To address this, we implemented a custom GEMM operator that accepts BF16 inputs and performs conversion and computation within registers. This approach maintains sufficient precision while significantly reducing computational and memory costs.

RMSNorm The following points require attention: The computation process should use FP32; The epsilon value must be consistent; Keep residual computation in FP32, and un-fuse the RMSNorm and residual in both training and inference.

RoPE Subtle implementation differences between training and inference should be carefully checked. For example, minor discrepancies often exist between a common PyTorch implementation and a RoPE operator in an inference engine, which can lead to slightly different outputs even with identical inputs.

Attention First, the used backend must be consistent between training and inference, e.g., FlashAttention (Dao et al., 2022). Second, watch out for the misalignment between prefill (used during training) and decode (used during inference). This issue causes the training-inference disparity to worsen with longer outputs, making RL training more prone to collapse.

MoE In addition to maintaining high precision in router computation, the non-stable `torch.topk` function must be replaced with a stable implementation. Furthermore, a deterministic order for token permutation and summation is required to prevent discrepancies, and MoE operators must be consistent between training and inference.

5.2.2 From the Algorithmic Perspective

When training-inference alignment is achieved at the systematic level, we observe that no additional algorithmic modifications are necessary.

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{x\sim\pi_{\text{rollout}}}\Big[\nabla_\theta \min\Big(\frac{\pi_{\text{training}}(x,\ \theta)}{\pi_{\text{rollout}}(x,\ \theta_{\text{old}})}\hat{A}, \text{clip}\Big(\frac{\pi_{\text{training}}(x,\ \theta)}{\pi_{\text{rollout}}(x,\ \theta_{\text{old}})}, 1-\epsilon,\ 1+\epsilon\Big)\hat{A}\Big)\Big]. \quad (5)$$

Equation 5 described above represents the ideal PPO optimization method, where the true rollout sampling distribution is used for importance sampling weighting. However, due to training-inference disparity, rollout probabilities often differ significantly from training probabilities, making them unsuitable for importance sampling. Most RL frameworks, e.g., verl (Sheng et al., 2024) and OpenRLHF (Hu et al., 2024a), resort to recomputing the training probabilities by re-forwarding the rollout data through the training engine, which leads to the following Equation 6.

$$\nabla_\theta \mathcal{J}'(\theta) = \mathbb{E}_{x\sim\pi_{\text{rollout}}}\Big[\nabla_\theta \min\Big(\frac{\pi_{\text{training}}(x,\ \theta)}{\pi_{\text{training}}(x,\ \theta_{\text{old}})}\hat{A}, \text{clip}\Big(\frac{\pi_{\text{training}}(x,\ \theta)}{\pi_{\text{training}}(x,\ \theta_{\text{old}})}, 1-\epsilon,\ 1+\epsilon\Big)\hat{A}\Big)\Big]. \quad (6)$$

However, Equation 6 is biased because it totally ignores the training-inference disparity. Consequently, the most reliable path forward is to use the rollout probabilities directly, as in Equation 5, but only after the training-inference disparity has been systematically corrected.

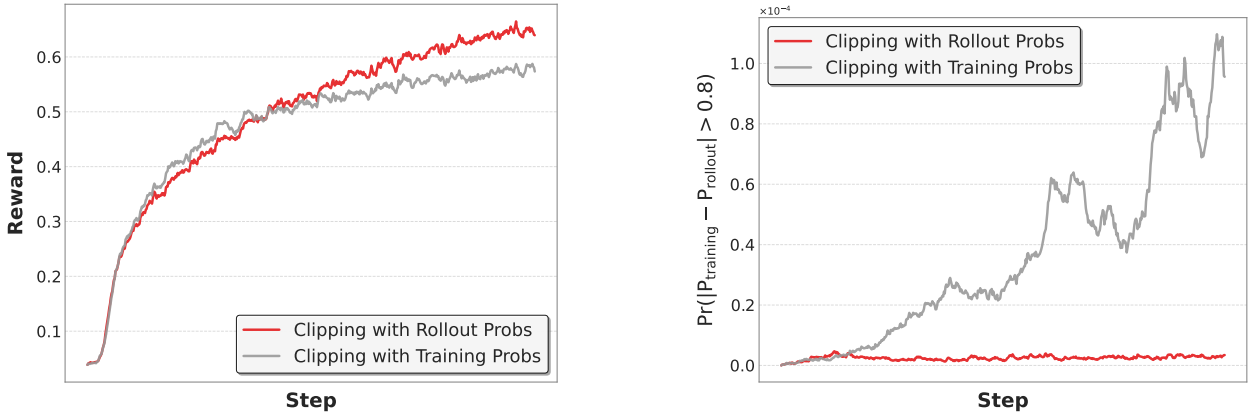
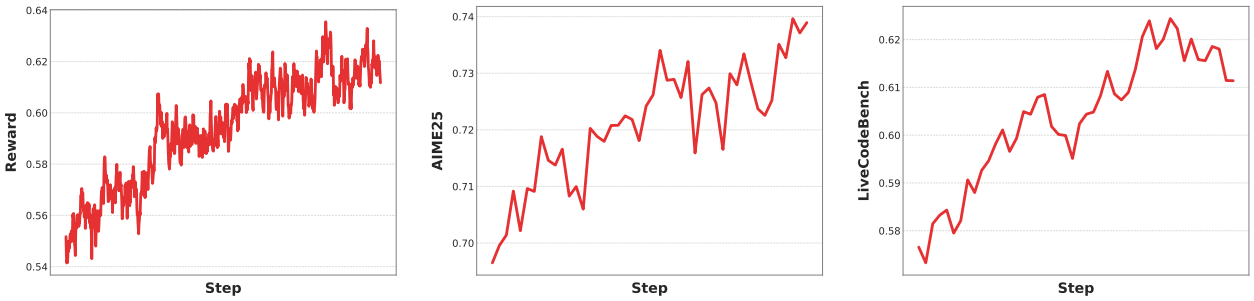


Figure 12 Comparison of PPO clip using rollout probabilities versus training probabilities. (Left) Reward. (Right) Proportion of tokens with an absolute training-inference probability difference greater than 0.8.

After systematic training-inference alignment, using rollout probabilities instead of recomputed training probabilities yields higher rewards in the later stages of training and maintains the training-inference disparity within a more stable range (as shown in Figure 12). In other words, under the premise of training-inference alignment, directly using rollout probabilities not only saves the time required for recomputing training probabilities but also further enhances the efficiency and stability of RL training.

5.2.3 Experimental Results of Reinforcement Learning

Based on the aforementioned improvements, the model demonstrates consistent growth in both training reward and test score throughout the RL process, as shown in Figure 13.



(a) Training Reward. **(b)** Test Score on AIME'25. **(c)** Test Score on LCB (sampled).

Figure 13 RL training curves of Ring-mini-linear-2.0.

6 Evaluation

We conduct a comprehensive evaluation of our Ring-linear series models, focusing on assessing the reasoning capabilities across multiple dimensions.

6.1 Evaluation Setups

This assessment is performed on a suite of 17 benchmarks across multiple reasoning dimensions:

Table 2 Performance of Ring-flash-mini-2.0 versus counterparts on various reasoning benchmarks.

	Benchmark	Ring-Mini-Linear-2.0	Ring-Mini-2.0	Qwen3-8B-Thinking	GPT-OSS-20B-Medium
Mathematical Reasoning	AIME'24	79.95	79.69	79.27	77.86
	AIME'25	73.65	74.06	71.25	73.85
	OlympiadBench	82.91	82.91	82.27	80.59
	CNMO'24	77.60	76.91	75.09	75.87
	LiveMathBench	83.64	83.98	82.92	83.40
	TheoremQA	69.69	70.09	68.81	66.81
Agent & Coding	Humaneval+	91.16	91.84	84.76	87.27
	MBPP+	79.70	80.79	79.63	79.86
	LiveCodeBench	59.53	62.56	56.94	54.90
	CodeForces(Elo)	83.84	84.80	73.31	82.25
	Spider	79.18	77.64	79.41	78.81
	BFCL_Live	73.99	74.26	75.99	54.64
General Reasoning	GPQA-Diamond	65.69	68.24	62.00	65.53
	SciBench	5.46	4.74	4.39	4.43
	DROP	83.20	88.55	87.13	76.49
	MuSR	77.21	75.99	76.92	76.53
	Multi_LogiEval	73.49	73.73	77.08	60.80

- **Mathematical Reasoning**: AIME'24 (MAA, 2024), AIME'25 (MAA, 2025), OlympiadBench (He et al., 2024), CNMO'24 (CMNO, 2024), LiveMathBench (Liu et al., 2024c), TheoremQA (Chen et al., 2023).
- **Agent and Coding**: Humaneval+ (Liu et al., 2024b), MBPP+ (Liu et al., 2024b), LiveCodeBench (Jain et al., 2024), CodeForces (Codeforces, 2024), Spider (Yu et al., 2018), BFCL-Live (Yan et al., 2024).
- **General Reasoning**: GPQA-Diamond (Rein et al., 2024), SciBench (Wang et al., 2023), DROP (Dua et al., 2019), MuSR (Sprague et al., 2023), Multi-LogiEval (Patel et al., 2024).

We benchmark our Ring-linear series models against state-of-the-art counterparts of a similar parameter scale. Specifically, Ring-mini-linear-2.0 is compared against Ring-mini-2.0 [13], Qwen3-8B-thinking (Yang et al., 2025), and GPT-OSS-20B-Medium (Agarwal et al., 2025). For Ring-flash-linear-2.0, the comparison includes Ring-Flash-2.0 [14], Qwen3-32B-Thinking (Yang et al., 2025), Gemini-2.5-Flash (Comanici et al., 2025), GPT-OSS-120B-Medium (Agarwal et al., 2025), Seed-OSS-36B-Instruct [15], and Qwen3-Next-80BA3B-Thinking [16].

6.2 Main Results

The performance results of Ring-mini-linear-2.0 and its comparison methods are summarized in Table 2. It is evident from the table that the Ring-mini-linear-2.0 model, despite its compact scale of only 1.6B activated parameters, demonstrates performance that is comparable to its counterpart models on various reasoning tasks.

Table 3 summarizes the benchmark results of our Ring-flash-linear-2.0 model alongside several key counterparts. The results indicate that despite its compact design, Ring-flash-linear-2.0 delivers performance that is highly comparable to these compared models across a diverse range of reasoning dimensions, demonstrating the model's comprehensive and robust reasoning capabilities.

[13] https://huggingface.co/inclusionAI/Ring-mini-2.0
[14] https://huggingface.co/inclusionAI/Ring-flash-2.0
[15] https://huggingface.co/ByteDance-Seed/Seed-OSS-36B-Instruct
[16] https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Thinking

Table 3 Performance of Ring-flash-linear-2.0 versus counterparts on various reasoning benchmarks.

	Benchmark	Ring-Flash-Linear-2.0	Ring-Flash-2.0	Qwen3-32B-Thinking	Gemini-2.5-Flash	GPT-OSS-120B-Medium	Seed-OSS-36B-Instruct	Qwen3-Next-80BA3B-Thinking
Mathematical Reasoning	AIME'24	90.73	91.04	82.66	80.36	80.40	91.70	92.08
	AIME'25	86.51	86.98	75.47	72.00	80.00	84.70	87.80
	OlympiadBench	87.36	88.10	84.69	83.41	82.32	87.06	87.80
	CNMO'24	84.98	85.07	78.21	82.38	79.95	91.75	85.33
	LiveMathBench	88.11	87.84	86.85	85.86	85.89	88.70	88.08
	TheoremQA	74.16	74.91	73.88	72.25	72.16	77.16	75.97
Agent & Coding	Humaneval+	91.84	92.00	90.24	91.77	84.83	91.23	91.77
	MBPP+	81.02	81.28	82.28	80.42	79.99	80.89	81.55
	LiveCodeBench	70.37	70.76	62.33	61.40	66.46	69.01	71.97
	CodeForces(Elo)	90.24	90.23	84.25	81.59	89.67	83.78	89.77
	Spider	80.86	81.70	81.00	77.27	78.71	78.16	82.95
	BFCL-Live	75.51	75.22	76.34	75.26	60.22	61.05	77.17
General Reasoning	GPQA-Diamond	74.49	75.25	68.40	82.80	73.10	71.40	77.20
	SciBench	5.13	5.18	4.34	4.11	4.93	4.11	4.68
	DROP	89.66	83.88	86.52	84.16	72.90	91.17	92.05
	MuSR	84.05	85.11	78.21	84.98	82.57	82.06	81.03
	Multi_LogiEval	75.21	76.18	77.01	77.97	73.96	77.90	72.62

7 Conclusion

This technical report presents models based on the Ring-linear hybrid architecture, specifically Ring-mini-linear-2.0 and Ring-flash-linear-2.0. By integrating a linear attention mechanism with high-performance FP8 fused kernels–LingHe, we have significantly enhanced both the training and inference efficiency of the models. Moreover, a well-aligned RL system has contributed to more stable training, further elevating the model's performance ceiling. However, we also recognize certain limitations in the current models. For instance, to maintain model effectiveness, the linear attention module maintains an identical attention head count for Q, K, and V, which brings heavy memory overhead. Furthermore, the remaining softmax attention modules introduce additional computational bottlenecks, impacting the overall efficiency of the hybrid architecture. Moving forward, we will continue to explore more efficient model architectures to better balance performance and efficiency.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

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.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. Theoremqa: A theorem-driven question answering dataset. *arXiv preprint arXiv:2305.12524*, 2023.

CMNO. Chinese national mathematics olympiad. https://www.cms.org.cn/Home/comp/comp/cid/12.html, 2024.

Codeforces. Codeforces competitive programming platform. https://codeforces.com, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

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

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024a.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. Minicpm: Unveiling the potential of small language models with scalable training strategies, 2024b. https://arxiv.org/abs/2404.06395.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models, 2024. https://arxiv.org/abs/2403.08763.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Jiawei Liu, Songrun Xie, Junhao Wang, Yuxiang Wei, Yifeng Ding, and Lingming Zhang. Evaluating language models for efficient code generation. In *First Conference on Language Modeling*, 2024b. https://openreview.net/forum?id=IBCBMeAhmC.

Junnan Liu, Hongwei Liu, Linchen Xiao, Ziyi Wang, Kuikun Liu, Songyang Gao, Wenwei Zhang, Songyang Zhang, and Kai Chen. Are your llms capable of stable reasoning? *arXiv preprint arXiv:2412.13147*, 2024c.

MAA. American invitational mathematics examination (aime) 2024. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime, 2024.

MAA. American invitational mathematics examination (aime) 2025. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models. *arXiv preprint arXiv:2406.17169*, 2024.

Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, et al. Transnormerllm: A faster and better large language model with improved transnormer. *arXiv preprint arXiv:2307.14995*, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

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.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. *arXiv preprint arXiv:2310.16049*, 2023.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. 2025a. https://arxiv.org/abs/2507.17702.

Changxin Tian, Jiapeng Wang, Qian Zhao, Kunlong Chen, Jia Liu, Ziqi Liu, Jiaxin Mao, Wayne Xin Zhao, Zhiqiang Zhang, and Jun Zhou. Wsm: Decay-free learning rate schedule via checkpoint merging for llm pre-training, 2025b. https://arxiv.org/abs/2507.17634.

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.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*, 2023.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Berkeley function calling leaderboard, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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. *Advances in neural information processing systems*, 37:115491–115522, 2024.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training, 2025. https://fengyao.notion.site/off-policy-rl.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy, 2024. https://arxiv.org/abs/2312.12728.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. 2025. https://arxiv.org/abs/2507.18071.