KIMI K3: OPEN FRONTIER INTELLIGENCE

TECHNICAL REPORT OF KIMI K3

Kimi Team

ABSTRACT

We introduce Kimi K3, a 2.8T parameter Mixture-of-Experts model with 104 billion activated parameters, native vision capabilities, and a 1-million-token context window. Kimi K3 is built on Kimi Delta Attention [63] and Attention Residuals [57], which improve information flow across sequence length and model depth. Together with Stable LatentMoE, which effectively activates 16 of 896 routed experts per token, and refined training and data recipes, these advances yield an approximately 2.5× improvement in overall scaling efficiency over Kimi K2 [58]. Post-training highlights reinforcement learning across general, agentic, and coding domains and multiple reasoning-effort levels, enabling compositional generalization and robust long-horizon execution. At 2.8T scale, Kimi K3 is supported by infrastructure advances in multiple areas: algorithm–system co-design for KDA, perfectly balanced expert-parallel training with efficient memory management, million-token agentic RL with persistent rollout and sandbox states, and deployment innovations.

Extensive evaluations show that Kimi K3 achieves frontier-level performance across long-horizon coding, agentic, knowledge, reasoning, and vision tasks. While its overall performance still trails the most powerful proprietary models, namely Claude Fable 5 and GPT-5.6 Sol, Kimi K3 consistently outperforms other open and proprietary models evaluated in our suite. We release the full Kimi K3 model weights to facilitate future research and accelerate the broader deployment and adoption of frontier intelligence.[1]

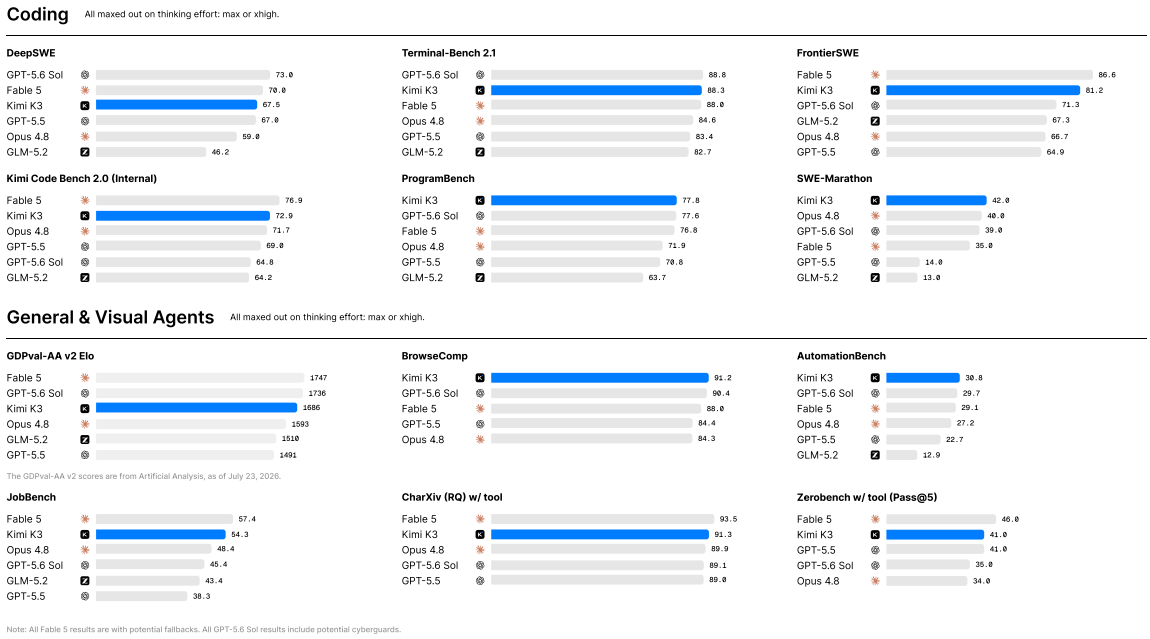


Figure 1: Kimi K3 main results.

[1] https://huggingface.co/moonshotai/Kimi-K3

1 Introduction

For much of the development of Large Language Models (LLMs), scaling meant investing more computation before deployment by training larger models on more data [54, 45]. The rise of reasoning models has established test-time computation as a second axis of scaling: OpenAI's o-series scales reinforcement learning and test-time reasoning [84, 83]; Anthropic's extended-thinking models allocate adaptive thinking budgets and interleave reasoning with tool use [6, 7]; DeepSeek-R1 [40] and Kimi K1.5 [118] show that large-scale reinforcement learning can elicit sophisticated reasoning behaviors from strong pre-trained models; and Kimi K2.5 Agent Swarm [59] further extends test-time scaling from sequential reasoning to parallel agent coordination. These advances have made test-time scaling a central focus of frontier research. However, while the open-source model ecosystem has advanced rapidly on the second axis, it has progressed slowly on the first: many recent models remain within or slightly above the 1T-class parameter regime [145, 29, 135, 120]. As increasingly sophisticated reasoning and agentic reinforcement learning methods are applied to pre-trained foundations of similar scale, open-source progress risks converging while the gap to the strongest proprietary systems widens. With Kimi K3, we pursue both scaling axes together to the frontier: scaling the pre-trained foundation to unprecedented 3T-class parameters while scaling reinforcement learning, reasoning effort, and long-horizon interaction at 1M context length.

We introduce Kimi K3, a native multimodal Mixture-of-Experts model with 2.8 trillion total parameters, 104 billion activated parameters, and a context window of up to one million tokens. Its architecture scales information flow across sequence length, network depth, and model width. Kimi Delta Attention (KDA) [63] provides efficient long-sequence mixing, with periodically interleaved Gated MLA layers preserving global interaction. Attention Residuals (AttnRes) [57] allows each layer to selectively attend to representations from all preceding layers. Stable LatentMoE expands the routed expert space to 896 experts, with 16 activated per token, while normalization, SiTU-GLU, and Quantile Balancing stabilize optimization at extreme sparsity. These architectural advances, combined with refined data and training recipes, yield an approximately $2.5\times$ improvement in overall scaling efficiency over Kimi K2 [58].

We pair this pre-training foundation with post-training designed explicitly for 1M context test-time scaling. Kimi K3 undergoes reinforcement learning across long-horizon coding, general agents, general reasoning and knowledge tasks, each spanning multiple reasoning-effort levels. Training environments include verifiable search and professional knowledge work, software engineering and kernel optimization, multimodal reasoning with vision-in-the-loop tool use, persistent assistant workflows, web development, and autonomous execution tasks. These environments train a general loop of reasoning, acting, observing, verifying, and adapting, often over hundreds or thousands of tool calls and millions of accumulated context tokens. Domain- and effort-specialized policies are consolidated into a unified model through multi-teacher on-policy distillation [75, 134, 29].

Realizing this regime requires infrastructure that scales with architecture complexity, model size, and trajectory length. For systems co-design for KDA, we develop fused kernels, KDA Context Parallelism, and state-aware prefix caching to make KDA efficient within devices, across devices, and across requests. For 2.8T-parameter MoE pre-training, MoonEP provides perfectly balanced expert execution with static computation shapes and zero-copy communication, while memory efficient training and multimodal encoder optimizations sustain utilization within bounded memory. For million-token agentic RL, our co-located system combines partial rollouts, external KV-cache retention, adaptive throttling and resumable microVM sandboxes to preserve long-lived model and environment state. Finally, specialized kernels, and cache- and budget-aware fleet scheduling translate these innovations into predictable production serving.

The resulting model establishes a new open frontier. On benchmarks spanning long-horizon coding, agentic, knowledge, reasoning, and vision tasks, Kimi K3 trails the strongest proprietary systems overall—Claude Fable 5 and GPT-5.6 Sol—and is consistently ahead of the other open and proprietary models evaluated in our suite, as shown in Fig. 1.

Our contributions are summarized as follows:

- **Pre-training at the open frontier.** We train a 2.8T-parameter native multimodal MoE model with 104B activated parameters and a 1M-token context window. KDA, AttnRes, Stable LatentMoE, refined data and training recipes collectively improve overall scaling efficiency by approximately $2.5\times$ over Kimi K2.
- **Reinforcement learning for multi-effort test-time scaling.** We conduct RL across general, agentic, and coding domains and multiple reasoning-effort levels, then consolidate the resulting capabilities into a unified model.
- **Infrastructure for multi-trillion-parameter, million-token intelligence.** We introduce KDA systems co-designs; MoonEP and memory-efficient infrastructure for 2.8T-parameter MoE pre-training; a co-located RL system with resumable sandboxes for million-token agentic trajectories; and more infrastructure innovations.
- **An open frontier model.** We release the full Kimi K3 model weights, making frontier intelligence available for research, deployment, and further innovation.

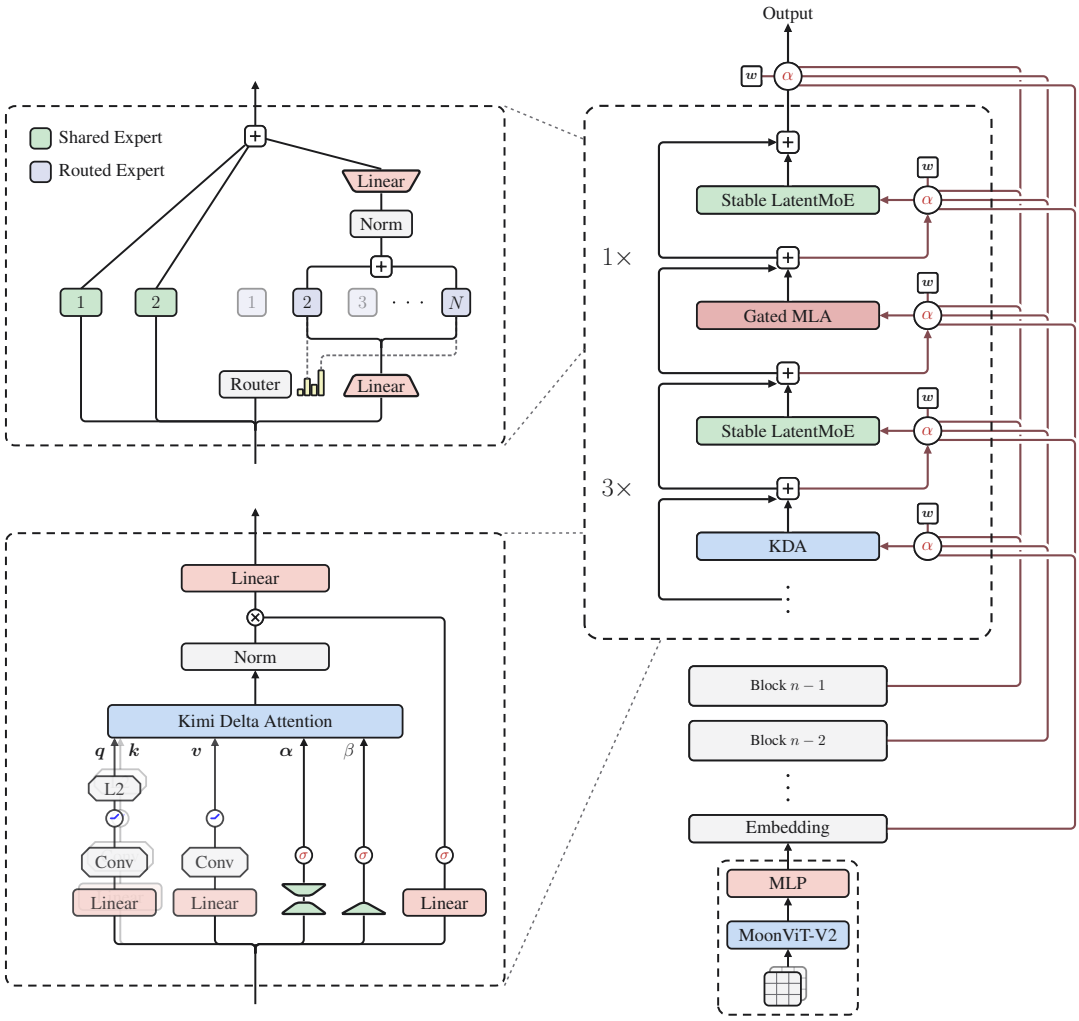


Figure 2: The Kimi K3 architecture, organized around token, channel, and layer mixing, with a native vision pathway at the input. Each block contains three Kimi Delta Attention (KDA) layers followed by one Gated MLA layer, with each attention layer paired with a Stable LatentMoE feed-forward network. Attention Residuals (AttnRes) use learned pseudo-queries ($\boldsymbol{w}$) to derive attention weights (α) over the embedding and preceding block outputs, enabling selective information flow across depth. **Top left**: the Stable LatentMoE module with shared and routed experts. **Bottom left**: the KDA module. **Bottom right**: the native vision pathway.

2 Model Architecture

The Kimi K3 architecture is designed to scale information flow along three complementary dimensions: sequence length, network depth, and model width. Along the sequence dimension, Hybrid Attention combines three Kimi Delta Attention (KDA) [63] layers with one Gated MLA layer in each block, providing an efficient mechanism for long-context token mixing while retaining selective high-capacity attention (§2.1). Along the depth dimension, Attention Residuals (AttnRes) [57] enable each module to selectively retrieve representations from the embedding, the current block, and preceding blocks, extending information access beyond conventional sequential residual accumulation (§2.2). Along the width dimension, each attention layer is followed by a Stable LatentMoE layer that performs sparse channel mixing, effectively activating 16 of 896 routed experts for each token (§2.3). For native vision, MoonViT-V2 encodes images and videos, and a lightweight projector maps the resulting visual features into the shared embedding space before backbone processing (§2.4). Together with Per-Head Muon (§2.5), these components provide a unified architecture for scaling information flow across tokens, layers, and channels. Combined with refined training and data recipes, they yield an approximately 2.5× improvement in overall scaling efficiency over Kimi K2. Figure 2 provides an overview of the architecture.

2.1 Hybrid Attention

Kimi K3 uses a layerwise hybrid of linear and global attention, combining KDA [63] with Gated MLA. Each block contains 3 KDA layers followed by 1 Gated MLA layer, giving a 3:1 mixing ratio. This pattern is repeated throughout the backbone. The two attention mechanisms are described separately below. An additional Gated MLA layer is placed at the end of the backbone, ensuring that the final layer always performs global attention.

2.1.1 Kimi Delta Attention

KDA extends the delta-rule recurrence [105, 138] with a channel-wise forget gate [63]. Consider a sequence of hidden states $\boldsymbol{x}_t \in \mathbb{R}^d$, where t indexes the token position and d is the model hidden dimension. For clarity, we first describe a single attention head, with query and key vectors $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^{d_k}$, value vector $\boldsymbol{v}_t \in \mathbb{R}^{d_v}$, and recurrent state $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$. KDA applies channel-wise decay before the delta-rule update:

$$\mathbf{S}_t = \left(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) \mathrm{Diag}(\boldsymbol{\alpha}_t) \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top, \qquad \tilde{\boldsymbol{o}}_t = \mathbf{S}_t^\top \boldsymbol{q}_t. \tag{1}$$

Here, $\boldsymbol{\alpha}_t \in (0,1)^{d_k}$ is the channel-wise one-step retention factor, and $\beta_t \in (0,1)$ controls the delta-rule write strength. Following Kimi Linear [63], KDA parameterizes the per-head quantities as

$$\begin{aligned}
\boldsymbol{q}_t^h, \boldsymbol{k}_t^h &= \mathrm{L_2Norm}\Big(\mathrm{Swish}\Big(\mathrm{ShortConv}\Big(\mathbf{W}_{q/k}^h \boldsymbol{x}_t\Big)\Big)\Big) \in \mathbb{R}^{d_k}, \\
\boldsymbol{v}_t^h &= \mathrm{Swish}\big(\mathrm{ShortConv}\big(\mathbf{W}_v^h \boldsymbol{x}_t\big)\big) \in \mathbb{R}^{d_v}, \\
\beta_t^h &= \mathrm{Sigmoid}\big(\mathbf{W}_\beta^h \boldsymbol{x}_t\big) \in (0,1), \\
\boldsymbol{z}_t^h &= \mathbf{W}_\alpha^\uparrow \mathbf{W}_\alpha^\downarrow \boldsymbol{x}_t + \boldsymbol{b}_\alpha^h \in \mathbb{R}^{d_k}.
\end{aligned} \tag{2}$$

The query, key, and value projections apply $\mathrm{ShortConv}$ followed by Swish [138], and the query and key are further normalized with $\mathrm{L_2Norm}$ [141]. The low-rank projection and head-specific bias $\boldsymbol{b}_\alpha^h \in \mathbb{R}^{d_k}$ produce a fine-grained decay logit $\boldsymbol{z}_t^h$ for each key channel. The lower-bounded mapping from $\boldsymbol{z}_t^h$ to $\boldsymbol{\alpha}_t^h$ is introduced after the chunkwise formulation below.

Chunkwise parallel form Following Kimi Linear [63], KDA is recurrent across chunks and parallel within each chunk. For a chunk size C, $\mathbf{X}_{[t]}$ stacks the token vectors in the t-th chunk for $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{U}, \mathbf{W}\}$. The matrix $\mathbf{S}_{[t]} \in \mathbb{R}^{d_k \times d_v}$ denotes the recurrent state entering chunk t. For positions $1 \le i \le j \le C$, define the channel-wise cumulative decay

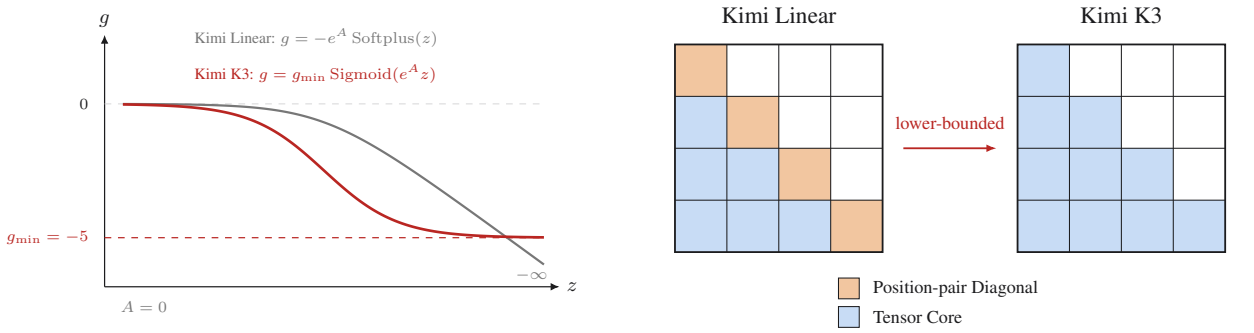
$$\boldsymbol{\gamma}_{[t]}^{i \to j} := \prod_{r=i}^{j} \boldsymbol{\alpha}_{[t]}^r, \qquad \boldsymbol{\gamma}_{[t]}^r := \boldsymbol{\gamma}_{[t]}^{1 \to r}. \tag{3}$$

As in Kimi Linear, $\boldsymbol{\Gamma}_{[t]}^{1 \to C} \in \mathbb{R}^{C \times d_k}$ stacks $\boldsymbol{\gamma}_{[t]}^1, \ldots, \boldsymbol{\gamma}_{[t]}^C$ row-wise. The UT transform produces $\mathbf{U}_{[t]}$ and $\mathbf{W}_{[t]}$, from which we define the pseudo-value term $\widetilde{\mathbf{V}}_{[t]} := \mathbf{U}_{[t]} - \mathbf{W}_{[t]} \mathbf{S}_{[t]}$. Given the incoming state $\mathbf{S}_{[t]}$, all outputs in chunk t are computed in parallel as

$$\begin{aligned}
\mathbf{A}_{[t]} &= \mathrm{Tril}\left[(\mathbf{Q}_{[t]} \odot \boldsymbol{\Gamma}_{[t]}^{1 \to C})(\mathbf{K}_{[t]} / \boldsymbol{\Gamma}_{[t]}^{1 \to C})^\top\right], \\
\mathbf{O}_{[t]} &= \underbrace{(\boldsymbol{\Gamma}_{[t]}^{1 \to C} \odot \mathbf{Q}_{[t]}) \mathbf{S}_{[t]}}_{\text{inter-chunk}} + \underbrace{\mathbf{A}_{[t]} \widetilde{\mathbf{V}}_{[t]}}_{\text{intra-chunk}}.
\end{aligned} \tag{4}$$

For a matrix $\mathbf{M}$, $\mathrm{Tril}(\mathbf{M})$ sets all strictly upper-triangular entries to zero and retains the lower-triangular entries, including the diagonal. This mask enforces causal interactions within the chunk, and the diagonal is retained because each output reads the state after the current-token update. The first term in $\mathbf{O}_{[t]}$ carries information from preceding chunks, whereas the second term accounts for interactions within the current chunk. We refer readers to Kimi Linear [63] for the UT transform and the full derivation of the chunkwise form.

Lower-bounded decay Eq. 4 rescales the keys in each chunk by the reciprocal cumulative decay $1/\boldsymbol{\Gamma}_{[t]}^{1 \to C}$. Because $\boldsymbol{\Gamma}_{[t]}^{1 \to C}$ is a product of retention factors in $(0,1)$, this reciprocal can grow without bound and overflow in finite precision [140, 63]. Kimi Linear controls this numerical range by computing relative decay in log space and dividing each chunk into secondary 16-token tiles [140, 63]. The off-diagonal tiles can then be computed with dense matrix multiplications on Tensor Cores directly. The diagonal tiles, in contrast, still require explicit position-pair computations, which remain the main intra-chunk bottleneck.



(a) Log-decay parameterization.

(b) Diagonal-tile computation.

Figure 3: Lower-bounded decay and its effect on chunkwise KDA computation. **(a)** Kimi Linear uses an unbounded negative-Softplus mapping, whereas Kimi K3 bounds the log-decay with a scaled sigmoid; the curves show $A = 0$ and $g_{\min} = -5$. **(b)** Kimi Linear evaluates each diagonal tile with an explicit position-pair computation, while the bounded range in Kimi K3 allows all causal tiles to use dense Tensor Core matrix multiplications.

Kimi K3 addresses this bottleneck by changing the mapping from the decay logits $\boldsymbol{z}_t^h$ to the per-step log-decay $\boldsymbol{g}_t^h$. Following GDN and Mamba-2, Kimi Linear uses the negative-Softplus mapping $\boldsymbol{g}_t^h = -e^{A_h}\,\mathrm{Softplus}(\boldsymbol{z}_t^h) \in (-\infty, 0)^{d_k}$ [138, 24, 63]. Kimi K3 instead uses a scaled sigmoid to bound the log-decay from below:

$$\begin{aligned} \boldsymbol{g}_t^h &= g_{\min}\,\mathrm{Sigmoid}\big(e^{A_h}\boldsymbol{z}_t^h\big) \in (g_{\min}, 0)^{d_k}, \\ \boldsymbol{\alpha}_t^h &= \exp(\boldsymbol{g}_t^h) \in (e^{g_{\min}}, 1)^{d_k}, \end{aligned} \tag{5}$$

where A_h is a learnable per-head log-scale and $g_{\min} = -5$ is fixed. We initialize $A_h = 0$, and each bias $\boldsymbol{b}_\alpha^h$ is initialized following [63, 24, 138]. With $g_{\min} = -5$, every retention factor satisfies $\alpha_{t,j}^h > e^{-5} \approx 6.7 \times 10^{-3}$, and the cumulative log-decay over a 16-token tile lies in $(-80, 0)$. The corresponding reciprocal rescaling factor is therefore smaller than e^{80} and remains within the BF16 dynamic range. This finite range allows both diagonal and off-diagonal tiles to use dense Tensor Core matrix multiplications, eliminating the position-pair diagonal path. This parameterization is closely related to the lower-bounded recurrence gates in prior work [97, 27, 91]. Fig. 3 illustrates the change in decay parameterization and its computational consequence.

Full-rank gate Finally, Kimi K3 changes KDA's output gate from the low-rank parameterization used by Kimi Linear [63] to an input-dependent full-rank projection. After applying head-wise RMSNorm [146] to the recurrent output, KDA applies data-dependent output gating [99]:

$$\boldsymbol{y}_t = \mathbf{W}_o[\mathrm{Sigmoid}(\mathbf{W}_g\boldsymbol{x}_t) \odot \mathrm{RMSNorm}(\tilde{\boldsymbol{o}}_t)]\,. \tag{6}$$

2.1.2 Gated MLA

Multi-head Latent Attention (MLA), introduced in DeepSeek-V2 [28], compresses the key–value representation of each token into a low-dimensional latent vector $\boldsymbol{c}_t = \mathbf{W}_c\boldsymbol{x}_t$. Instead of caching full head-specific keys and values, MLA caches $\boldsymbol{c}_t$ and reconstructs the content keys and values through learned up-projections during attention computation. This factorization reduces the KV-cache footprint while retaining global token-to-token attention. MLA was subsequently adopted by Kimi K2 and Kimi K2.5 [58, 59], and Kimi K3 retains it in the periodic global-attention layers.

Unlike Kimi K2 and Kimi K2.5, Kimi K3 follows the hybrid design of Kimi Linear [63] and applies No Position Encoding (NoPE) to all MLA layers. Consequently, no explicit positional encoding is applied to their queries or keys. The intervening KDA layers provide position-sensitive and recency-aware sequence mixing, while the MLA layers provide unrestricted global content interaction. This separation also avoids modifying positional-encoding parameters when extending the context length, such as retuning a RoPE frequency base or applying YaRN [92].

In addition, Kimi K3 augments MLA with an input-dependent, channel-wise full-rank output gate. Let $\tilde{\boldsymbol{o}}_t$ denote the ungated MLA output at position t; the gated output is

$$\boldsymbol{y}_t = \mathbf{W}_o[\mathrm{Sigmoid}(\mathbf{W}_g\boldsymbol{x}_t) \odot \tilde{\boldsymbol{o}}_t]\,. \tag{7}$$

The gate projection $\mathbf{W}_g$ is full rank, matching the new parameterization used by KDA in Kimi K3. This gate allows each token to modulate the channels read from global attention [99].

To correct the biased rounding error that arises in flash attention, we adopt the method of [98] and keep the attention output in FP32 during training. This choice doubles the on-chip footprint of the output tile; we therefore redesign the training kernel to overlap it with the KV staging buffers instead of the query tile, freeing shared memory for a deeper KV pipeline and higher training throughput.

2.2 Attention Residuals

Standard residual connections [43] compress all prior information into a single state $\boldsymbol{h}_l$ over depth — a bottleneck reminiscent of RNNs over time. For sequence modeling, the Transformer replaced recurrence with attention [10, 125], allowing each position to selectively access all previous positions with data-dependent weights. Attention Residuals (AttnRes) [57] applies the same methodology to depth: each layer selectively retrieves representations from all preceding layers rather than accumulating them uniformly.

Full Attention Residuals For each layer l, we define a layer-specific learnable pseudo-query $\boldsymbol{q}_l = \boldsymbol{w}_l \in \mathbb{R}^d$ and keys and values

$$\boldsymbol{k}_i = \boldsymbol{v}_i = \begin{cases} \boldsymbol{h}_1 & i = 0 \\ f_i(\boldsymbol{h}_i) & 1 \le i \le l-1 \end{cases} \tag{8}$$

where $f_i(\boldsymbol{h}_i)$ is the output of layer i and $\boldsymbol{h}_1$ is the token embedding. The attention weights follow a softmax kernel $\phi(\boldsymbol{q}, \boldsymbol{k}) = \exp\left(\boldsymbol{q}^\top \mathrm{RMSNorm}(\boldsymbol{k})\right)$ [55, 146], where the RMSNorm prevents layers with large-magnitude outputs from dominating the weights:

$$\alpha_{i\to l} = \frac{\phi\left(\boldsymbol{q}_l, \boldsymbol{k}_i\right)}{\sum_{j=0}^{l-1} \phi\left(\boldsymbol{q}_l, \boldsymbol{k}_j\right)}, \qquad \boldsymbol{h}_l = \sum_{i=0}^{l-1} \alpha_{i\to l} \cdot \boldsymbol{v}_i. \tag{9}$$

Since network depth is modest ($L < 100$), the $O(L^2 d)$ arithmetic of this *full* form is affordable; the practical overhead is the $O(Ld)$ memory (and cross-stage communication under pipeline parallelism) for keeping all layer outputs alive.

Block Attention Residuals To reduce this overhead, we partition the L layers into N blocks of $S = L/N$ layers each. Within block n (layer indices $\mathcal{B}_n$), layer outputs are reduced to a single representation by summation, $\boldsymbol{b}_n = \sum_{j\in\mathcal{B}_n} f_j(\boldsymbol{h}_j)$, with $\boldsymbol{b}_n^i$ denoting the partial sum over the first i layers of the block; we set $\boldsymbol{b}_0 = \boldsymbol{h}_1$ so the token embedding is always included as a source. Across blocks, full attention is applied over only the N block-level representations: for the i-th layer in block n, the value matrix is

$$\mathbf{V} = \begin{cases} [\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}]^\top & \text{if } i = 1 \text{ (first layer of block } n) \\ [\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{b}_n^{i-1}]^\top & \text{if } i \ge 2 \text{ (subsequent layers)} \end{cases} \tag{10}$$

with keys and attention weights following Eq. 8 and Eq. 9. The final output layer then aggregates all N block representations. Under Block AttnRes, memory and communication overhead drop from $O(Ld)$ to $O(Nd)$, while this block structure also bounds the inference-time state, enabling the parallel inter-block results to be better merged with the sequential intra-block partial sums via online softmax [79], significantly reducing inference time cost.

Empirically, $N \approx 8$ recovers most of the benefit across model scales [57]; for Kimi K3, we partition its layers into 8 blocks with 12-layer size, giving a partial final block and 9 total blocks when counting the embedding layer.

2.3 Stable LatentMoE

Increasing both the expert pool and the number of active experts expands the space of expert specializations, but in a conventional MoE each selected expert receives the full d-dimensional token representation, so communication and expert-weight traffic grow with the routing multiplicity. LatentMoE [32] makes this expansion affordable by separating the full model width from the routed-expert width: shared experts retain a full-width path for common transformations, whereas specialized routed experts operate in a compact latent space of width ℓ. This enables Kimi K3 to scale channel mixing to 896 routed experts with 16 active experts per token, corresponding to a sparsity of 56.

This extreme sparsity amplifies two failure modes of the vanilla design. First, the routed path composes $\mathbf{W}^\downarrow$, a gated multi-branch expert feed-forward network, and $\mathbf{W}^\uparrow$ into a chain of nearly four consecutive matrix multiplications. This ill-conditioned structure, combined with the 2.8-trillion-parameter scale, produces exploding internal activations in the routed branch. Second, balancing the load of nearly 10^3 experts exceeds the regime in which existing auxiliary-loss-free

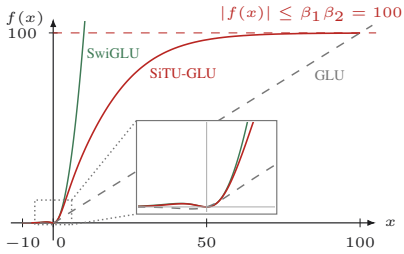
	Gate branch	Up branch	Curve
GLU [26]	$\sigma(x)$	x	
SwiGLU [107]	$x \cdot \sigma(x)$	x	
SiTU-GLU	$\beta_1 \tanh\left(\frac{x}{\beta_1}\right) \cdot \sigma(x)$	$\beta_2 \tanh\left(\frac{x}{\beta_2}\right)$	

Figure 4: Gate and up branches of GLU, SwiGLU, and SiTU-GLU, together with their scalar responses, where σ denotes the sigmoid function. Both branches receive the scalar input x, and all curves share the domain $x \in [-10, 100]$; the inset magnifies the near-origin region. SiTU-GLU, shown in red with $\beta_1 = 4$ and $\beta_2 = 25$, closely follows SwiGLU near the origin and approaches the bound $|f(x)| \leq \beta_1\beta_2 = 100$ for large positive inputs, whereas SwiGLU remains unbounded.

bias updates remain well behaved. Stable LatentMoE addresses these two failure modes with three components: an RMSNorm before the up-projection and Sigmoid Tanh Unit GLU (SiTU-GLU) to suppress activation explosion, and Quantile Balancing (QB) for load balancing.

As illustrated in Fig. 2, the layer follows the shared- and routed-expert organization of DeepSeekMoE [23]. For $\boldsymbol{x} \in \mathbb{R}^d$, the shared experts process $\boldsymbol{x}$ directly, while the routed path projects it to $\boldsymbol{z} = \mathbf{W}^{\downarrow}\boldsymbol{x} \in \mathbb{R}^{\ell}$, dispatches $\boldsymbol{z}$ to the selected experts, and maps their weighted aggregate back to $\mathbb{R}^d$ through $\mathbf{W}^{\uparrow}$:

$$\begin{aligned} \boldsymbol{u} &= \sum_{i \in \mathcal{T}_k(\boldsymbol{x})} p_i E_i^{\text{routed}}(\mathbf{W}^{\downarrow}\boldsymbol{x}), \\ \boldsymbol{y} &= \sum_{j=1}^{N_s} E_j^{\text{shared}}(\boldsymbol{x}) + \mathbf{W}^{\uparrow}\,\text{RMSNorm}(\boldsymbol{u}). \end{aligned} \tag{11}$$

Here, $\boldsymbol{u} \in \mathbb{R}^{\ell}$ is the aggregated routed representation, $E_j^{\text{shared}} : \mathbb{R}^d \to \mathbb{R}^d$ and $E_i^{\text{routed}} : \mathbb{R}^{\ell} \to \mathbb{R}^{\ell}$ are the shared and routed expert feed-forward networks, and p_i is the router weight defined by the Quantile Balancing rule below. Kimi K3 fixes the number of full-width shared experts to $N_s = 2$ in every layer.

2.3.1 Normalized LatentMoE

The original LatentMoE directly applies $\mathbf{W}^{\uparrow}$ to the aggregated routed representation $\boldsymbol{u}$, whose scale can vary with the selected experts and their routing weights. As shown in Eq. 11, Kimi K3 instead inserts RMSNorm [146] between expert aggregation and the up-projection. This normalization reduces the sensitivity of the routed branch to scale variation before it is combined with the full-width shared branch. Beyond stabilizing training, the additional RMSNorm consistently improves validation loss and downstream benchmarks.

2.3.2 Sigmoid Tanh Unit GLU

Gated Linear Units (GLUs) modulate a linear value branch with a sigmoid-activated gate, computing $\text{Sigmoid}(\mathbf{W}_g\boldsymbol{x}) \odot \mathbf{W}_u\boldsymbol{x}$ [26]. SwiGLU replaces the sigmoid gate with $\text{Swish}(x) = x\,\text{Sigmoid}(x)$ and yields strong empirical performance in Transformers [107]. SwiGLU has subsequently become a widely adopted FFN design in large language models, while a complete account of its empirical effectiveness remains open.

However, both multiplicative factors in SwiGLU are unbounded, so coincident large coordinates can produce activation outliers and increase overflow risk in low-precision arithmetic. The sigmoid gate of the original GLU avoids unbounded gate growth, but it does not retain the approximately linear positive regime of Swish. This motivates an activation that controls large-value growth while preserving the characteristic local and positive-side response of SwiGLU. Other recent efforts have explored alternative parameterizations of this trade-off [51].

To satisfy these requirements, we propose Sigmoid Tanh Unit GLU (SiTU-GLU). SiTU-GLU applies the smooth cap $\text{softcap}(x, \beta) = \beta \tanh(x/\beta)$ to the linear factor of the Swish gate and independently to the up branch:

$$\text{SiTU-GLU}(\boldsymbol{x}) = \left[\beta_1 \tanh\left(\frac{\mathbf{W}_g\boldsymbol{x}}{\beta_1}\right) \odot \text{Sigmoid}(\mathbf{W}_g\boldsymbol{x})\right] \odot \left[\beta_2 \tanh\left(\frac{\mathbf{W}_u\boldsymbol{x}}{\beta_2}\right)\right], \tag{12}$$

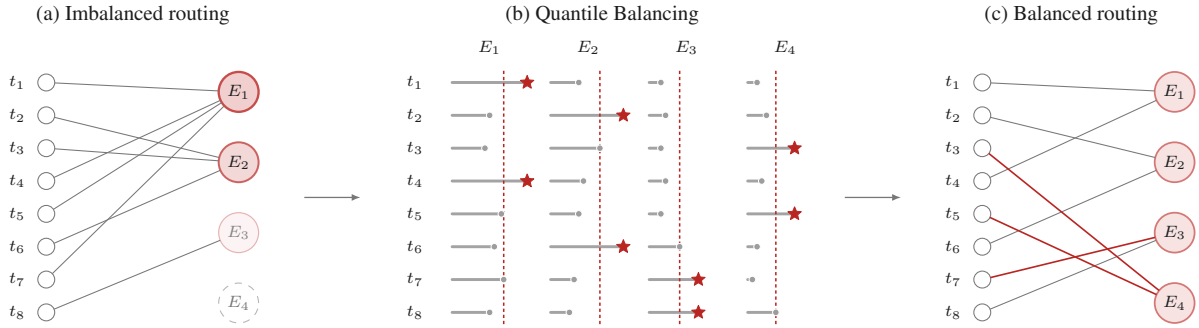


Figure 5: Illustration of Quantile Balancing with $m = 8$ tokens, $n = 4$ routed experts, and $k = 1$ selected expert per token. (a) Token-wise Top-k routing (tokens on the left, experts on the right) produces loads $(4, 3, 1, 0)$; darker circles indicate overheated experts, whereas faded and dashed circles indicate underutilized and dying experts, respectively. (b) Each gray bar is the margin of the currently biased score, $s_{i,j} + b_j^{(t)} - \alpha_i^{(t)}$, so the row-wise maxima reproduce the routing in (a). The dashed red line in each column is the bias adjustment $b_j^{(t)} - \widehat{b}_j^{(t+1)}$, placed at the $(q{+}1)$-th largest margin so that exactly $q = 2$ margins exceed it. The marker ★ denotes the row-wise Top-k choice after subtracting the column adjustments, i.e., the routing in (c). (c) The retained choices yield the balanced load $(2, 2, 2, 2)$; red edges denote assignments changed by QB.

For Kimi K3, we set the soft-cap hyperparameters to $\beta_1 = 4$ for the gate branch and $\beta_2 = 25$ for the up branch. The scaled $\tanh$ is approximately linear near the origin and bounded at large magnitude, allowing SiTU-GLU to preserve the local response of SwiGLU while controlling both factors in the product. Fig. 4 compares the branch definitions and scalar responses of GLU, SwiGLU, and SiTU-GLU on a common slice.

§ B gives the local expansion, limiting case, formal output bound, and comparison with hard clamping.

2.3.3 Quantile Balancing

Unlike auxiliary-loss-based routing [33], Kimi K3 adopts auxiliary-loss-free routing [30]. Load balancing is implemented by adding an expert-specific bias b_j to the router score used for Top-k selection. For token $\boldsymbol{x}_i$, the router computes $\boldsymbol{s}_i = \mathrm{Sigmoid}(\mathbf{W}_r \boldsymbol{x}_i)$ and applies

$$\mathcal{T}_i = \mathrm{argtop}_k(\boldsymbol{s}_i + \boldsymbol{b}), \qquad p_{i,j} = \frac{s_{i,j}}{\sum_{r \in \mathcal{T}_i} s_{i,r}}, \quad j \in \mathcal{T}_i. \tag{13}$$

Because $\boldsymbol{b}$ is omitted from $p_{i,j}$, it regulates dispatch without altering the mixture weights or the gradient-based optimization of the router. The original method updates $\boldsymbol{b}$ with the fixed-step rule $b_j^{(t+1)} = b_j^{(t)} + \gamma\,\mathrm{sign}(\bar{\ell} - \ell_j^{(t)})$ [30], for which γ trades off slow adaptation against load oscillation. Maintaining balanced loads becomes more challenging as LatentMoE increases the routed expert pool to 896 per layer. Imbalanced routing slows expert-parallel training and may leave some experts poorly trained [47].

To address this limitation, we introduce Quantile Balancing (QB), which sets each expert bias from the router-score quantile that matches its target load [111]. Consider a training batch of m tokens routed to n experts with Top-k selection, so the target load is $q := mk/n$ tokens per expert. QB derives the next bias from a single forward pass. Routing replaces the Top-k selection with Top-$(k{+}1)$ on the biased score $\boldsymbol{s}_i + \boldsymbol{b}^{(t)}$: the first k entries are the routes actually taken, while the $(k{+}1)$-th entry is the cutoff $\alpha_i^{(t)}$ that an expert must exceed to enter token i's Top-k. Taking the cutoff from Top-$(k{+}1)$ routing avoids a separate token-side quantile. We then choose each expert bias so that expert j receives its target load: with the cutoffs fixed, the token count routed to expert j under a candidate bias $\widehat{b}_j^{(t+1)}$ is

$$\sum_{i=1}^{m} \mathbf{1}\left[s_{i,j} + \widehat{b}_j^{(t+1)} > \alpha_i^{(t)}\right],$$

which is monotonically decreasing in the threshold $-\widehat{b}_j^{(t+1)}$. Assuming no ties, setting this count to q makes $-\widehat{b}_j^{(t+1)}$ the $(q{+}1)$-th largest margin $s_{i,j} - \alpha_i^{(t)}$, so that exactly q margins stay above the threshold. Since $q/m = k/n$, this is

the $(1 - k/n)$-quantile of the margins across tokens, giving the QB update

$$\widehat{b}_j^{(t+1)} \leftarrow -\operatorname{quantile}_{1-k/n}\left(\boldsymbol{s}_{:,j} - \boldsymbol{\alpha}^{(t)}\right),$$
$$\boldsymbol{b}^{(t+1)} \leftarrow \widehat{\boldsymbol{b}}^{(t+1)} - \operatorname{mean}\left(\widehat{\boldsymbol{b}}^{(t+1)}\right)\mathbf{1}. \tag{14}$$

The margins subtract the biased cutoff $\alpha_i^{(t)}$ from the raw score $s_{i,j}$, so the old bias enters the update only through the cutoffs, and the second line removes a common offset that leaves Top-k selection unchanged. For causality, the update takes effect only in the next step [30], i.e., a batch is never routed with a bias derived from itself. Fig. 5 illustrates the case $m = 8$, $n = 4$, and $k = 1$, where each expert receives the target load $q = 2$. The final bias is frozen at inference. The balanced-assignment derivation is given in § C.

Histogram estimation At scale, the quantile in Eq. 14 spans the full global batch, whose margins number in the millions and are spread across ranks and accumulation steps, so gathering them for an exact quantile is not viable at training time. We instead read each expert's quantile from a histogram of its margins: a single all-reduce sums the per-rank bin counts, and the quantile is recovered from the pooled counts. Because counts are additive, the histogram represents the pooled global batch regardless of how tokens are sharded, so the estimate reflects the whole-batch quantile up to the bin width, at a communication cost of only a few hundred bins per expert. This histogram estimator is the method we use in practice; we give more detailed descriptions of it and its error bound in § D.

2.4 Native Vision

Kimi K3 is natively multimodal: text, images, and videos are processed by a single shared backbone within one context, with no post-hoc modality-alignment stage. This design is the architectural foundation of the long-horizon, vision-in-the-loop behavior described in §1. Rendered outputs and the code that produced them live in the same token stream, the model can write code, inspect screenshots or video frames of the result, and iteratively refine visual artifacts—user interfaces, graphics, video—with no cross-model hand-off.

MoonViT-V2 A key departure from Kimi K2.5 is that we train Kimi K3 vision encoder, *MoonViT-V2, entirely from scratch with next-token prediction*. Prior practice, including Kimi K2.5 itself, initializes the vision encoder from a contrastively pre-trained model such as SigLIP, under the premise that pre-trained visual knowledge gives the model a head start. We depart from this practice primarily for training stability. When a pre-trained encoder is attached to the LLM, joint optimization becomes unstable: the SigLIP-initialized MoonViT-3D shows persistently higher gradient norms with frequent spikes, while MoonViT-V2 remains stable throughout training (Fig. 6). Training with next-token prediction also allows the encoder's representations to be shaped directly by the language-modeling objective, rather than by a contrastive loss that favors global semantics over fine-grained textual and structural cues. Notably, we find MoonViT-V2 matches the SigLIP-initialized baseline across vision evaluations, indicating that contrastive pre-training is unnecessary as an initialization for multimodal language models at scale.

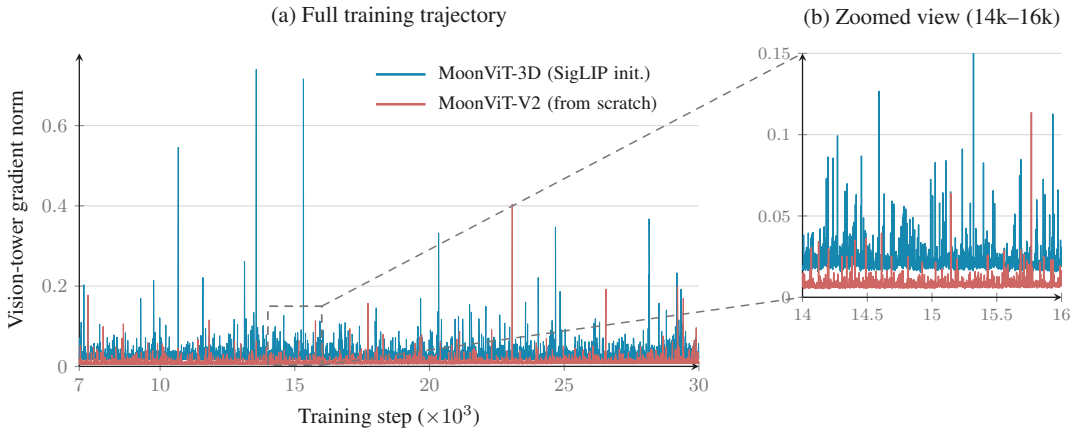


Figure 6: Vision-tower gradient norms in our pre-training ablations. Compared with the SigLIP-initialized MoonViT-3D, the from-scratch MoonViT-V2 maintains lower gradient norms with fewer spikes, indicating more stable optimization.

Architecture This training recipe builds on a vision pathway that follows the overall design of Kimi K2.5 [59, 61]: visual inputs are first encoded by MoonViT-V2 and then mapped by a lightweight MLP projector into the LLM. MoonViT-V2 is a 27-layer vision transformer with roughly 0.4B parameters that adopts RMSNorm and removes all bias terms from its linear and attention projections, a design that further stabilizes the from-scratch optimization above. Images and videos are processed with fully shared parameters, as in MoonViT-3D: attention is factorized into intra-frame spatial and inter-frame temporal passes, and temporal pooling further compresses tokens along the time dimension. Before projection, a pixel-shuffle operation with 2×2 downsampling reduces the number of visual tokens by a factor of four, keeping inputs of up to 3584×3584 pixels affordable within the 1M-token context.

2.5 Per-Head Muon

Following Kimi K2, Kimi K3 adopts Muon [53] as the optimizer for its matrix parameters. For attention projections, we further refine it into a per-head variant: instead of applying Newton–Schulz orthogonalization to the full Q, K, and V projection matrices, we partition their momentum matrices along the head dimension and orthogonalize each head's block separately. The intuition is that full-matrix orthogonalization treats all heads as a single coupled block, so heads with larger gradient or momentum scales dominate the shared update direction, while smaller-scale heads receive insufficiently normalized updates; per-head orthogonalization equalizes the update scale across heads. In practice, this design yields more balanced learning dynamics across heads and improves training stability at larger scales. It also slightly reduces optimizer overhead, as Newton–Schulz iterations on tall per-head blocks are cheaper than on the full projection matrix.

3 Pre-Training

3.1 Pre-Training Data

Kimi K3 is pre-trained on a curated corpus spanning four primary text domains—Web Text, Code, Mathematics, and Knowledge—together with a large-scale vision corpus. The vision data covers captions, interleaved image–text documents, OCR, perception, video, and visual coding data. Our data pipelines build on those developed for Kimi K2 [58] and refined in Kimi K2.5 [59].

Text data Each domain is filtered by a combination of rule-based heuristics, classifier-based quality scoring, and deduplication, with domain-specific sampling rates determined by ablation studies on smaller models. Following the rephrasing recipe of Kimi K2 [58], we rephrase knowledge and mathematics corpora with style and perspective-diverse prompting, chunk-wise autoregressive generation, and fidelity verification against the source documents.

Vision data The vision corpus follows the taxonomy of Kimi K2.5 [59], combining open-source collections with in-house pipelines for filtering, synthesis, and deduplication. During training, coordinate supervision is provided in both absolute and normalized ([0,1]) formats, enabling precise and resolution-robust localization. In addition to classical text-captioned images, we substantially scale up programmatic multimodal data, coupling code snippets with their rendered visuals across domain-specific formats including SVG, 3D assets, Webpage, Game, and CAD schematics.

3.2 Scaling Law

Taken together, the architectural, data, and training improvements described in the previous sections define our new model family. Since these changes also alter the optimal training regime, we conduct dedicated scaling-law studies to retune key hyperparameters, including the batch size, learning rate, tokens-per-parameter ratio (TPP) and the model shape. Evaluated on held-out OOD validation data, the scaling law curves in (Fig. 7) show that these improvements collectively deliver an approximately $2.5\times$ gain in overall scaling efficiency over Kimi K2. Table 1 provides a detailed architectural comparison between Kimi K2 and Kimi K3, highlighting the structural changes that contribute to this improvement.

Our scaling-law study consistently favors cosine decay over Warmup Stable Decay (WSD) [46], leading us to adopt cosine decay as the default learning rate schedule. We compare cosine decay and WSD under a fixed minimum learning rate. Although prior work has reported that WSD can match or even outperform cosine decay, we observe that the two schedules exhibit markedly different optimal hyperparameters. Even under the same model size and training-token budget, their optimal peak learning rates and batch sizes differ substantially. As a result, comparing the two schedules using a shared set of hyperparameters may unfairly favor one simply because those hyperparameters are better aligned with it. To ensure a fair comparison, we conduct an independent scaling-law search for each schedule. Under their respective optimal hyperparameter settings, cosine decay consistently achieves a lower final loss than WSD.

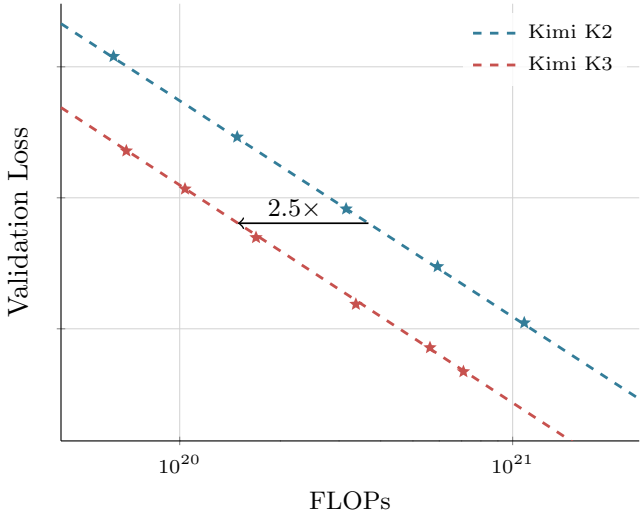


Figure 7: Fitted scaling-law curves for Kimi K2 and Kimi K3. Kimi K3 achieves $2.5\times$ gain in scaling efficiency over Kimi K2.

Table 1: Architectural comparison between Kimi K2 and Kimi K3.

	Kimi K2	**Kimi K3**	**Δ**
Architecture	MoE	MoE	–
#Layers	61	93	↑ 52%
Total Parameters	1.04T	2.78T	↑ 167%
Activated Parameters	32.6B	104.2B	↑ 220%
Hidden Dimension	7,168	7,168	=
Latent MoE Dimension	–	3584 (0.5×)	–
MoE Hidden Dimension per Expert	2,048	3,072	↑ 50%
Routed Experts	384	896	↑ 133%
Experts Active per Token	8	16	↑ 100%
Shared Experts	1	2	↑ 100%
Attention Heads	64	96	↑ 50%
Number of Dense Layers	1	1	=
Vocabulary Size	160K	160K	=
Training Context Length	128K	1M	8×
Attention Mechanism	MLA	Hybrid KDA–MLA	–
Activation Function	SwiGLU	SiTU-GLU	–
Attention-Layer Composition	61 MLA	69 KDA + 24 MLA	–
Number of MTP Layers	1 layer	1 layer	=
Total Parameters of ViT	-	401M	-
#ViT Layers	-	27 layers	-
Patch Size of ViT	-	14	-
#Attention Heads of ViT	-	12	-

3.3 Training Recipe

Kimi K3 adopts a native multimodal training strategy in which language and vision are jointly optimized from the start of training, rather than grafting a vision encoder onto a pre-trained language model through a post-hoc alignment stage. Under this paradigm, visual and textual tokens are interleaved within a single next-token prediction objective, enabling the shared backbone to learn unified multimodal representations from the outset.

We optimize the model using the Per-Head Muon optimizer (§ 2.5) together with the weight-clipping mechanism introduced in Kimi K2, while adopting QB (§ 2.3.3) for MoE load balancing. We use a cosine learning rate schedule with a 1% linear warmup. Weight decay is set to 0.1 throughout.

Our pre-training begins with a context length of 8k tokens, which is later extended to 64k tokens in a subsequent training phase.

3.4 Long-Context Extension

Positional encoding Kimi K3 uses no explicit positional embedding (NoPE), and instead encodes positional information implicitly through the recurrent gating and decay mechanism of KDA. As a result, the model extrapolates directly to 1M-token contexts without any positional-encoding modification, such as RoPE rescaling or interpolation [92].

Long-context data Long documents and videos from natural sources contain a substantial amount of low-quality content, including near-duplicates, binary blobs, truncated files, video clips, and invalid machine-generated logs. We therefore process them through a dedicated cleaning pipeline that combines exact and fuzzy deduplication, supplemented by perceptual hashing over frames for video, together with heuristic and classifier-based quality filtering, and structural validation. Because genuinely long and coherent documents and videos are scarce relative to short text, we upsample them so that the long-context distribution is not overwhelmed by short sequences during cooldown. Length alone, however, does not confer long-range capability. To address this, we synthesize additional long-context data by carefully permuting and concatenating multimodal documents and sub-tasks, so that the embedded tasks can be solved only by attending to information scattered across the full 1M-token context. This trains the attention mechanism at the intended scale and prevents it from degenerating into local patterns.

Progressive context extension Kimi K3 supports a context window of up to 1 million tokens. We achieve this through extending the context window progressively as training proceeds, following a four-stage curriculum. The window grows from 8K to 64K tokens during pre-training, and from 256K to 1M tokens during the cooldown phase. Concentrating the costly long-sequence computation within a small fraction of the overall training budget keeps the curriculum economical while still allowing the model to adapt gradually to increasingly long-range dependencies. The sequence-dimension partitioning that makes million-token training tractable for the KDA layers is described in §5.1.2.

4 Post-Training

4.1 Method

Our post-training pipeline follows a three-stage paradigm: initializing baseline agent capabilities via supervised fine-tuning (SFT), developing specialized domain experts at varying reasoning effort via Reinforcement Learning (RL), and consolidating these domain-specific policies into a single model using Multi-Teacher On-Policy Distillation (MOPD).

4.1.1 Supervised Fine-Tuning

The SFT stage establishes a high-quality cold-start policy for the subsequent RL stage. Building on the SFT pipeline of previous Kimi models [58, 59], we expand the SFT dataset for Kimi K3, substantially broadening its coverage of complex agentic tasks. Specifically, we synthesize data trajectories using domain-specialized models from the prior Kimi series, followed by multi-stage verification and human-in-the-loop annotation. To represent these complex agentic trajectories consistently, we serialize all data with our XTML-based chat template (eXtensible Token Markup Language; see § F for details). Collectively, these steps yield a large-scale instruction dataset that endows Kimi K3 with adaptive reasoning, precise tool calling, and robust execution in long-horizon agentic scenarios. In addition, we apply quantization-aware training (QAT) from the SFT stage onward, with MXFP4 weights and MXFP8 activations (§ 4.1.4).

4.1.2 Reinforcement Learning

While SFT provides a solid cold-start foundation, RL is critical to unlocking higher-order reasoning and execution capabilities. Rather than training specialized RL models for individual tasks, we scale RL across three broad domains, each encompassing a wide spectrum of sub-tasks, and train a single expert for each domain at every reasoning effort level: (i) *general tasks*, spanning general experience, vision, reasoning, faithfulness, search capabilities, and knowledge work tasks; (ii) *general agents*, spanning long-horizon assistant tasks, deep research, and paragraph-level writing; and (iii) *coding agents*, spanning software engineering (SWE), coding experience, kernel tasks, and web development. As shown in Figure 8, scaling RL FLOPs consistently improves a variety of capabilities across knowledge, reasoning, vision, general agent, and coding. Crossing these three domain experts with three reasoning effort levels in $\{\text{low}, \text{high}, \text{max}\}$ yields a total of nine expert models.

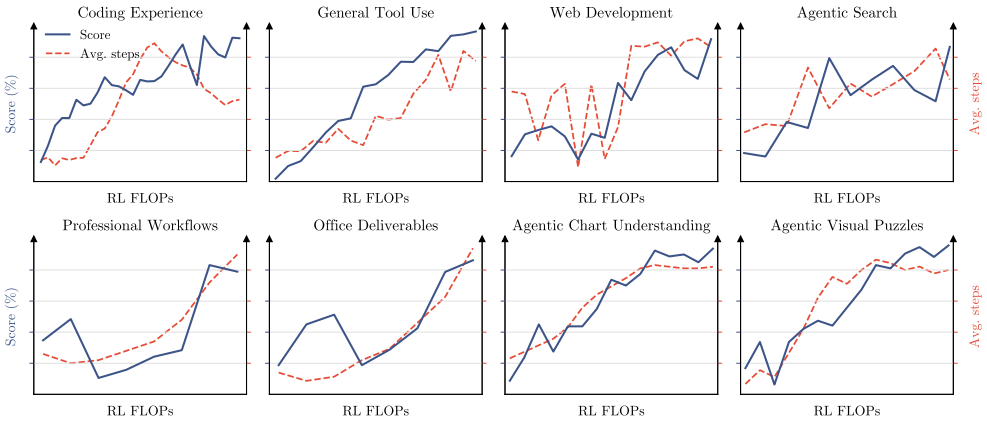


Figure 8: Scores and the average assistant steps across a variety of public and in-house evaluations during RL. By scaling RL FLOPs, tool-call steps scale up consistently, accompanied by a comprehensive improvement in the model's overall capability.

Algorithm To mitigate the long-tail latency that intensifies in long-horizon tasks, we extend the *partial rollout* scheme from our synchronous RL framework [118, 59]. During the rollout phase of each iteration, we sample K completions for each of N prompts, maintaining an active workload of $N \times K$ trajectories. Rather than waiting for all rollouts to terminate, the generation phase pauses as soon as a fraction $\lambda \in (0, 1)$ of trajectories completes (i.e., λNK), allowing policy optimization to proceed without execution stragglers. Paused rollouts are enqueued and prioritized for resumption at the start of the next iteration, powered by our sandbox infrastructure (§ 5.3.2). Once all K responses for a prompt complete, they are immediately dispatched for policy optimization, which follows the algorithm in Kimi K2.5 [59]. Under our partial rollout scheme, an individual long-horizon trajectory naturally spans multiple iterations, introducing data staleness that threatens training stability. Our policy optimization algorithm inherently tolerates such an extreme off-policy regime through a per-token regularization. By constraining policy updates within a localized neighborhood, this regularization enables the algorithm to robustly handle highly stale data and sustains training stability.

Reasoning Effort RL To fine-tune reasoning effort while maximizing token efficiency, we implement a per-problem budget control mechanism during RL [59]. We associate each problem x with an initial token budget $b_0(x)$ estimated from the cold-start model, and override the task reward with -1 for trajectories whose total token budget $T(y)$ exceeds a scaled threshold $\tau \cdot b_0(x)$. For general tasks, $T(y)$ measures the number of thinking tokens, whereas for agentic tasks, $T(y)$ accounts for the cumulative output tokens, including both reasoning traces and tool-call arguments. Training follows a stage-wise curriculum over the budget multiplier τ. We first train a *max-budget* variant with a relatively large τ, while still capping the maximum budget to suppress excessive overthinking. We then anneal τ to smaller values to obtain the *high*- and *low*-effort expert models. The adjustment of τ is configured per domain under human-in-the-loop guidance. Trajectories produced by the resulting experts at all reasoning levels are jointly collected for supervised fine-tuning and multi-teacher on-policy distillation.

Agentic Generative Reward Model For non-verifiable general tasks, we adopt an Agentic Generative Reward Model (GRM), retaining the tournament-style group reward with binary comparisons as in Kimi K2.5 [58, 59]. Beyond generic agentic capabilities for enhanced judgment, the agentic judge is required to follow a mandatory protocol: (1) read the outcome, product, or text output; (2) generate a rubric; (3) score each candidate against the rubric; and (4) record the rubric-assigned scores in a scorepad. To mitigate reward hacking toward increasingly verbose outputs, we apply a budget-based verbosity control analogous to the reasoning-effort control above: given an initial verbosity ℓ_0 estimated from the cold-start model and a multiplier σ, a candidate whose output length exceeds $\sigma \cdot \ell_0$ automatically loses the binary comparison.

4.1.3 Multi-Teacher On-Policy Distillation

We adopt Multi-Teacher On-Policy Distillation (MOPD) to consolidate these domain-specialized capabilities across varying reasoning efforts into a unified model [75, 134, 29]. During training, for a given domain d and a sampled reasoning effort level $e \in \{\text{low}, \text{high}, \text{max}\}$, optimization is guided by the corresponding teacher model $\pi_{\text{teacher}}^{(d,e)}$ among the nine experts. Given an input query x and the prefix response $y_{<t}$, the per-token OPD reward evaluated on y_t between

the teacher $\pi^{(d,e)}_{\text{teacher}}$ and the student π_θ is defined as:

$$r^d_{\text{opd}}(y_t \mid e, x, y_{<t}) = \text{clip}\left(\text{sg}\left(\log \frac{\pi^{(d,e)}_{\text{teacher}}(y_t \mid x, y_{<t})}{\pi_\theta(y_t \mid e, x, y_{<t})}\right), -R_{\max}, R_{\max}\right), \tag{15}$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operator, and $R_{\max} > 0$ is a clipping threshold to constrain extreme advantage signals, thereby stabilizing RL training. This dense reward signal seamlessly integrates into our RL framework, naturally enabling infrastructure-level optimizations such as partial rollout training for long-horizon tasks. While we also experimented with more fine-grained top-k distillation objectives, we observed no clear advantage in either convergence speed or final performance in our setting.

4.1.4 Deployment-Aware Post-Training

MXFP4 Quantization-Aware Post-Training To reduce memory footprint and serving cost at deployment, we quantize the MoE expert weights — which dominate the model's parameter memory — to MXFP4 [103], with activations computed in MXFP8, while all non-expert components (attention projections, latent MoE projections, shared experts, and MoE routers) remain in higher precision. We perform quantization-aware training (QAT) [49] throughout the entire post-training stage, covering both SFT and RL, so that the model adapts to quantization-induced precision loss. During RL, rollout and training share the same quantization scheme — eliminating the train–inference mismatch.

Draft Model Fine-Tuning Optimizing inference efficiency is crucial for serving complex, long-horizon agentic models. Kimi K3 is pre-trained with a multi-token-prediction (MTP) layer that mirrors the structure of a backbone block. As the draft model of EAGLE-3 [71] comprises a single decoder layer whose structure matches the MTP layer, we fine-tune the pre-trained MTP layer into an EAGLE-3-style draft model, with the target model frozen and only the draft layer and its feature-fusion projection updated. Following the training-time test protocol of EAGLE-3, the draft is unrolled for seven steps during training; beyond the first step, where the target-side features of the newest position are unavailable, the draft consumes its own outputs from earlier steps, mirroring the recurrent drafting procedure at inference.

The draft input fuses low-, mid-, and high-level features of the target model, taken from the outputs of the 1st, 4th, and final AttnRes blocks, respectively (§ 2.2). These features are concatenated and projected to the hidden size by a bias-free matrix $\boldsymbol{W}_{\text{E3}}$, initialized as $[\,\mathbf{0}\;\;\mathbf{0}\;\;\boldsymbol{I}\,]$ so that the fused representation coincides at initialization with the high-level feature $\boldsymbol{h}_h$ — the input on which the MTP layer was pre-trained — and gradually learns to incorporate the low- and mid-level features during fine-tuning.

The speedup of speculative decoding is governed by the per-token acceptance rate $\sum_{x\in\mathcal{V}} \min(p(x), q(x))$ under lossless speculative sampling, where p and q denote the next-token distributions of the target and draft models. Since minimizing the conventional KL-divergence surrogate does not guarantee maximizing this rate for a capacity-limited draft model, we directly optimize the likelihood-based LK loss [104], the negative logarithm of the acceptance rate itself,

$$\mathcal{L}_{\text{LK}} = -\log \sum_{x\in\mathcal{V}} \min(p(x), q(x)), \tag{16}$$

with p and q evaluated at temperature 1 and no auxiliary ground-truth cross-entropy term. Draft fine-tuning follows the post-training QAT configuration (§ 4.1.4), with MoE expert weights in MXFP4 and their input activations in MXFP8, while non-expert modules remain in higher precision.

4.2 RL Task Synthesis and Agentic Environments

The effectiveness of our RL framework relies heavily on rich, diverse, and robustly verifiable environments. To support scalable training across complex long-horizon tasks, we design a series of specialized white-box environments and task synthesis paradigms.

4.2.1 Unified White-Box RL Environment

Training with a single fixed agent harness can cause a model to overfit to a particular tool schema, system prompt, context management mechanism, or interaction protocol. To address this, we develop a unified white-box RL environment that represents an agent harness as a collection of configurable, composable modules, including tool interfaces, system prompts, context management strategies, skills, memories, subagents, and other components. Composing these modules through configuration, the environment can instantiate mainstream harnesses such as Kimi Code [56], Claude Code [15], Codex [20], OpenClaw [86], and Hermes [44], as well as entirely new ones. During RL training, we dynamically

construct different harness configurations for different task groups, exposing Kimi K3 to diverse combinations of these modules rather than the conventions of any single harness. The same abstraction also readily supports RL across various task domains, providing a scalable foundation for training more general-purpose agents.

4.2.2 Knowledge-Graph-Guided Task Synthesis

Motivation and overview The quality and diversity of post-training tasks are largely determined by their source materials. Retrieval guided by fine-grained concepts surfaces specialized and underrepresented knowledge, while sampling across diverse concepts broadens domain coverage. To control both granularity and coverage at scale, we build a self-evolving, hierarchically organized knowledge graph that agents continuously expand through web-scale exploration across knowledge-intensive and coding domains. Figure 9 illustrates the task synthesis pipeline.

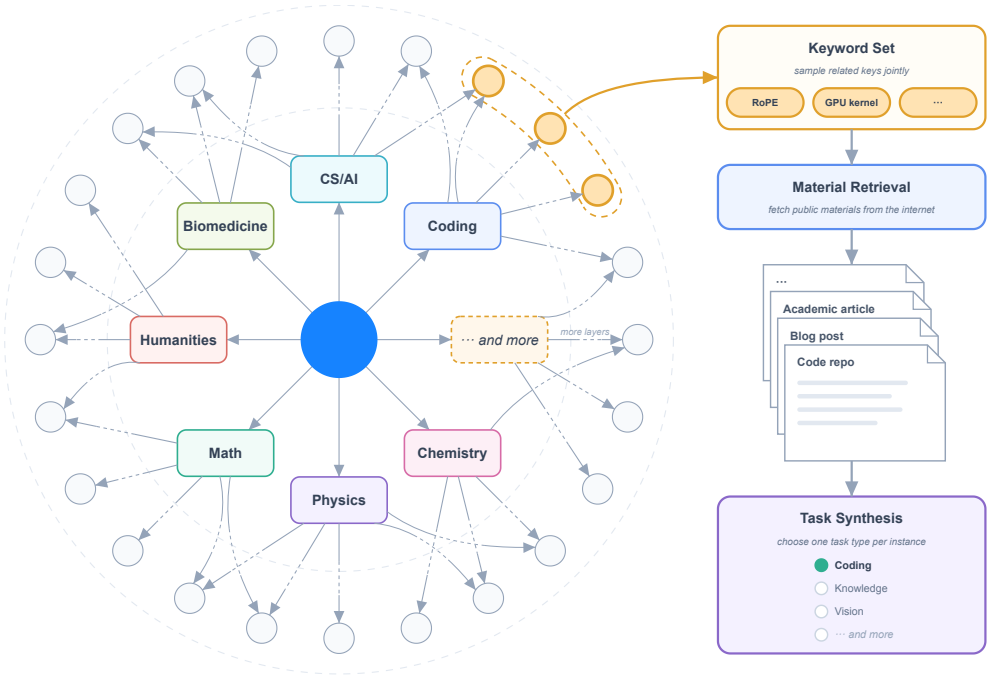


Figure 9: Overview of knowledge-graph-guided task synthesis. The hierarchically organized knowledge graph represents concepts at multiple levels, ranging from broad domains to fine-grained concepts. Related nodes are sampled to form a keyword set that guides the retrieval of publicly available source materials. For each synthesis instance, the system selects a task type and uses the retrieved materials to synthesize a corresponding task.

Agentic knowledge graph construction We construct the knowledge graph as a directed acyclic graph through recursive, agent-driven expansion. The expansion process begins with a predefined set of coarse-grained seed nodes. An agent instance is then assigned to each node and performs multiple web searches to investigate the corresponding concept. Before adding new nodes, the agent explores the existing graph to identify equivalent or related concepts, reuse existing nodes where appropriate, and minimize duplication. Edges are always directed from the coarser concept to the finer one, regardless of which endpoint the agent discovers first. Newly added nodes are subsequently assigned to agents for further exploration. A branch stops expanding when the assigned agent determines that the current concept is sufficiently atomic.

Material retrieval and task synthesis To target a desired distribution across domains and task types, the system samples nodes at varying levels of granularity, either individually or in related combinations. Keywords derived from the sampled nodes are combined with contextual information from their ancestors in the knowledge graph to formulate web queries. The retrieved real-world materials are assembled so that a synthesis agent produces training tasks of various task types.

4.2.3 Verifiable Problems in Agentic Environments

We train Kimi K3 on verifiable problems in agentic environments; representative examples include multi-step complex information searching, where the model plans its research, gathers evidence from the web step by step, and produces

a verifiable answer; the real day-to-day work of professionals, such as investment banking, data analysis, and legal practice, where the model decomposes a complex request, operates domain tools in a sandbox, and completes a deliverable over dozens to hundreds of steps; and multi-step verifiable visual reasoning over STEM problems, visual puzzles, and chart understanding. Each visual-reasoning trajectory is generated in an agent environment equipped with a Python interpreter in an isolated sandbox: the model iteratively writes and executes code to crop, zoom, or transform the input image, perform precise computation, or verify intermediate results, and receives the execution outputs — including generated images — as new observations over multiple interaction steps. As the model learns to perform more image operations and collect more observations, its performance on complex visual reasoning tasks steadily improves.

4.2.4 Kernel Optimization Tasks

To strengthen Kimi K3's GPU kernel optimization capabilities, we build a large-scale suite of kernel tasks ranging from single-operator kernels to fused mega-kernels, sourced from high-quality GitHub repositories such as Flash Linear Attention [139]. The suite spans diverse GPU programming approaches, such as CUDA, Triton, CuTe DSL, Gluon, ThunderKittens [110], and TileLang [129], and covers widely used GPU architectures and numerical formats including BF16, FP8, and FP4. Rewards evaluate both correctness and performance: each kernel provides a PyTorch reference implementation, and solutions exceeding a predefined numerical error threshold receive zero reward. Performance is scored against an expert implementation, where matching it yields a reward of 0.5 and approaching the hardware roofline increases the reward toward 1. To ensure that rewards reflect genuine optimization, we develop a hacking-detection system that penalizes reward-hacking strategies such as CUDA graph replay, input caching, and precision reduction, and we continuously extend it with new safeguards as new hacking strategies are observed during Kimi K3's development.

4.2.5 Personal Assistant Tasks

For long-horizon personal assistant tasks, we develop realistic mock implementations of widely used applications, such as Gmail, Notion, Slack, and Canvas. They preserve the core semantics of their real-world counterparts while enabling reproducible, large-scale interaction without external APIs or rate limits. Building on these mock applications, we design complex tasks inspired by real-world professional workflows in scenarios like human resources, legal services, and finance. In each task, the agent operates in a persistent, evolving environment over multiple simulated days and encounters dozens of interdependent events distributed across applications. A single rollout may involve up to thousands of tool calls and millions of context tokens. Each event carries its own evaluation criterion, assessed by deterministic rules or LLM-based evaluators. The initial workspace is constructed by agents that autonomously search the web for reference materials and transform them into a coherent, task-relevant environment. We also extend our RL framework to support such living environments, modeling complex event streams and the induced world-state transitions.

4.2.6 Autonomous Execution Tasks

We introduce Autonomous Execution Tasks (AET), an environment paradigm that trains long-horizon agent intelligence through verify-in-the-loop optimization. Each task specifies an initial state, a constrained goal, a tool-based action space, execution budgets, and an independent verifier. Agents see only the objective, context, constraints, and verification interfaces, without reference trajectories or predefined procedures, and must autonomously perform task decomposition, tool selection, planning, error recovery, and termination. Rewards are grounded in the verifier's evaluation of the final environment state rather than the agent's self-reported completion. We design multiple types of verifiers that support diverse environments, including black-box system replication (Figure 10), quantitative factor discovery, and tax auditing. In each environment, agents iteratively submit solutions, receive verifier feedback, and refine their strategies, training a general loop of hypothesizing, acting, analyzing feedback, and adapting. Reward hacking is mitigated by isolating agents from verifiers, pairing public verifiers that offer diagnostic feedback with hidden verifiers that evaluate held-out scenarios, and applying penalty-based rewards under limited submission budgets.

4.2.7 Web Development Tasks

We construct a diverse suite of expert-curated web development tasks covering typical scenarios. Inputs range from one-line scene descriptions to multi-paragraph specifications; artifacts span websites, interactive games, 3D/WebGL scenes, data visualization, SVGs, and full-stack applications. Every task runs in a containerized sandbox and is rolled out under diverse agent scaffolds rather than a single fixed harness, to promote cross-scaffold generalization. Rewards consist of two components: deterministic checks and model judging by an internal reward model. Deterministic checks functionally test application behavior, and score structural and pixel-level similarity for tasks that replicate a reference. The reward is zeroed when a project fails to build, runs with errors, or fakes rather than implements the artifact. Model judging uses other models to perform source code inspection or to look at and interact with the output artifact.

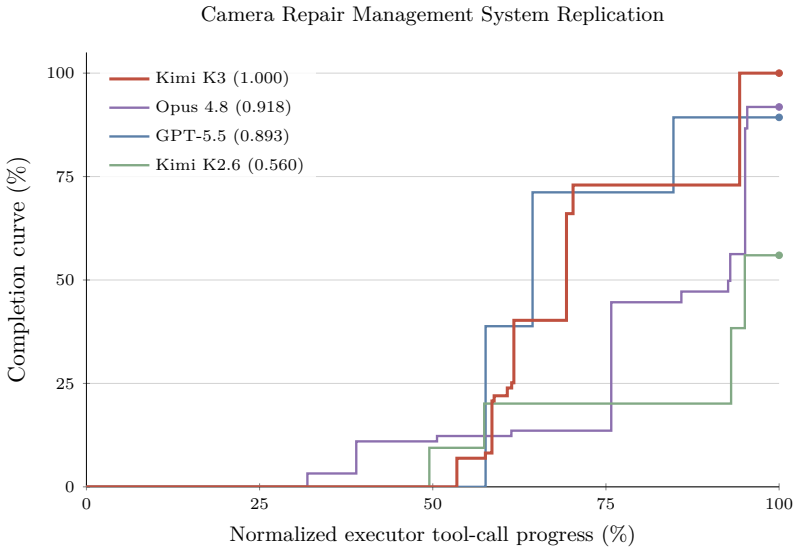


Figure 10: Completion curves on Camera Repair Management System, a black-box system replication task in which the agent reconstructs a hidden 3D-camera repair system as a web application through oracle queries. Completion denotes verifier-assessed task progress.

5 Infrastructure

Kimi K3 combines three system challenges rarely encountered in a single model: hybrid KDA attention, 3T-class sparse multimodal training and inference, and million-token agentic workloads. Our infrastructure is co-designed with these challenges across the model lifecycle. At the architecture level, high-performance KDA kernels and Context Parallelism make the recurrent formulation efficient within and across devices, in both training and inference. During pretraining, balanced expert execution, reduced memory footprint, and communication-overlapped scheduling sustain high utilization at scale. During 1M-token agentic RL, hierarchical state management and resumable sandbox execution preserve long trajectories across iterations. Finally, state-aware KDA prefix caching, specialized inference kernels, and cache- and budget-aware scheduling translate these efficiencies into predictable production serving.

5.1 Algorithm-System Co-Design for KDA

KDA replaces the growing key–value cache of softmax attention with a fixed-size recurrent state $\mathbf{S} \in \mathbb{R}^{d_k \times d_v}$ (§2.1.1), whose serial update poses challenges in parallel execution, in exchange for a fixed-size state that is cheap to transfer and reuse. The designs below address the first property and exploit the second at two levels of execution, with fused kernels within a device and KDA Context Parallelism across devices.

5.1.1 KDA Kernels across Regimes

The serial dependence of the KDA state is at odds with the GPU's preference for wide, uniform parallelism, and it manifests as a different bottleneck in each execution regime. We design a dedicated kernel for each regime.

Chunkwise kernel for training and prefill The chunkwise form of KDA is parallel within each chunk but serial across chunks, since the recurrent state must propagate from chunk to chunk. Executed naively, these two phases alternate, leaving the SMs idle during the serial propagation. We therefore develop FlashKDA [14], a CUTLASS-based chunkwise kernel that overlaps intra-chunk computation with cross-chunk state propagation. The kernel decomposes the work into token-parallel stages and a head-parallel recurrence, each scheduled and tuned independently, and substantially outperforms the Triton reference implementation. FlashKDA serves both training and inference prefill and is auto-dispatched as a backend of flash-linear-attention [139].

Intra-device context parallelism for long-context prefill Tensor parallelism partitions heads across devices but never shortens the recurrence, so under pure TP deployment, prefilling an ultra-long sequence leaves most SMs idle when each rank holds only a few heads. The key observation is that the state transition of each segment can be evaluated

independently of the incoming state and composed exactly afterward. An automatic SM-level context-parallel (CP) planner [142, 139] therefore partitions the sequence across the SMs of a single rank, evaluates the segment transitions in parallel, and merges them to recover each segment's exact initial state. In contrast to the cross-device KCP of §5.1.2, this parallelism is entirely intra-device and incurs no cross-device communication.

KDA decoding presents challenges distinct from those encountered during training and prefill. We discuss these challenges in detail in §5.4.2.

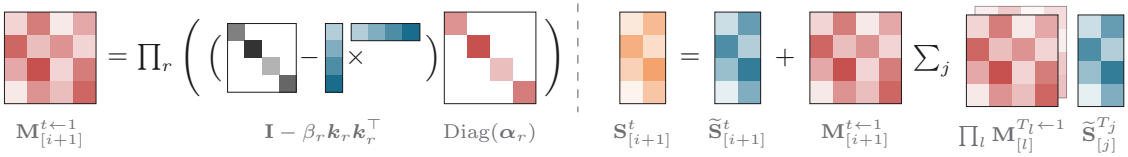
5.1.2 KDA Context Parallelism

The communication overhead of context parallelism differs fundamentally between softmax and linear attention. Softmax attention requires ranks to exchange key–value blocks whose size grows with the sequence length [72]. Linear attention instead carries the preceding context in a fixed-size recurrent state $\mathbf{S} \in \mathbb{R}^{d_k \times d_v}$. Prior context-parallel methods exploit the additive recurrence of vanilla linear attention by computing, on each rank, the state that the local tokens generate from $\mathbf{S} = \mathbf{0}$ and summing these local states over the preceding ranks to recover the incoming state [114, 113].

This direct summation, however, is insufficient for KDA. Recall from Eq. 1 that KDA updates its state as $\mathbf{S}_t = \mathbf{M}_t\mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$, where $\mathbf{M}_t := \left(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) \mathrm{Diag}(\boldsymbol{\alpha}_t)$. KDA's delta rule applies the token-dependent matrix $\mathbf{M}_t$ to the incoming state before adding the current write. Consequently, the effect of a local sequence segment depends on the state entering that segment and cannot be determined from the state computed with $\mathbf{S} = \mathbf{0}$ alone.

To preserve this dependence, we introduce KDA Context Parallelism (KCP), which decomposes the effect of each segment into two locally computable quantities, a cumulative transition acting on the incoming state and a state generated locally from zero. Following the chunkwise notation of §2.1.1, we write $\mathbf{S}_{[i]}^t$ for the recurrent state within the segment of rank i after t local tokens, so that $\mathbf{S}_{[i]}^{T_i}$ denotes the state leaving rank i and entering rank $i+1$. We write $\widetilde{\mathbf{S}}_{[i]}^t$ for the state of the same recurrence started instead from $\mathbf{S} = \mathbf{0}$. For an arbitrary state entering the $(i+1)$-th of P context-parallel ranks, the state after t local tokens is

$$\begin{aligned} \mathbf{M}_{[i+1]}^{t\leftarrow 1} := \prod_{r\leftarrow 1}^{t} \mathbf{M}_r \in \mathbb{R}^{d_k\times d_k}, \qquad \mathbf{S}_{[i+1]}^t &= \widetilde{\mathbf{S}}_{[i+1]}^t + \mathbf{M}_{[i+1]}^{t\leftarrow 1}\mathbf{S}_{[i]}^{T_i} \\ &= \widetilde{\mathbf{S}}_{[i+1]}^t + \mathbf{M}_{[i+1]}^{t\leftarrow 1} \sum_{j=1}^{i} \Big(\prod_{l\leftarrow j+1}^{i} \mathbf{M}_{[l]}^{T_l\leftarrow 1} \Big) \widetilde{\mathbf{S}}_{[j]}^{T_j} \in \mathbb{R}^{d_k\times d_v}. \end{aligned} \tag{17}$$



where $\mathbf{M}_{[i+1]}^{t\leftarrow 1}$ denotes the cumulative transition of the first t local tokens. The first term contains the state generated by the local tokens, whereas the second term propagates the context from preceding ranks through the local KDA updates. At $t = T_{i+1}$, both quantities $\mathbf{M}_{[i+1]}^{T_{i+1}\leftarrow 1}$ and $\widetilde{\mathbf{S}}_{[i+1]}^{T_{i+1}}$ can be computed using only the local tokens, before $\mathbf{S}_{[i]}^{T_i}$ is available, and are the fragments each rank exchanges with the others.

The summation in Eq. 17 shows that every state is composed purely from locally computed fragments. These rank-level updates compose associatively, so the incoming state of each rank can be recovered by a prefix scan [77]. Each rank first computes $\mathbf{M}_{[i]}^{T_i\leftarrow 1}$ and $\widetilde{\mathbf{S}}_{[i]}^{T_i}$ locally, then exchanges both tensors with one `all-gather` [139].[2] After the `all-gather`, rank $i+1$ reconstructs $\mathbf{S}_{[i]}^{T_i}$ by processing preceding fragments of the same document in order, starting from $\mathbf{S} = \mathbf{0}$ and applying $\mathbf{S} \leftarrow \mathbf{M}_{[j]}^{T_j\leftarrow 1}\mathbf{S} + \widetilde{\mathbf{S}}_{[j]}^{T_j}$ at each fragment. Therefore, KCP requires only a fixed-size `all-gather` for recurrent-state synchronization and achieves linear compute scaling.

5.2 Infra for 3T-class Pre-Training

Kimi K3 pre-training combines Pipeline Parallelism (PP) with virtual stages (VP) [48, 81], Expert Parallelism (EP) [66], ZeRO-1 Data Parallelism [100], Pipeline ZeRO-2 gradient sharding [145], and Context Parallelism (CP, §5.1.2) [50].

[2] The construction builds on DeltaNet context parallelism [142]. The KDA implementation is available in FLA PR #691.

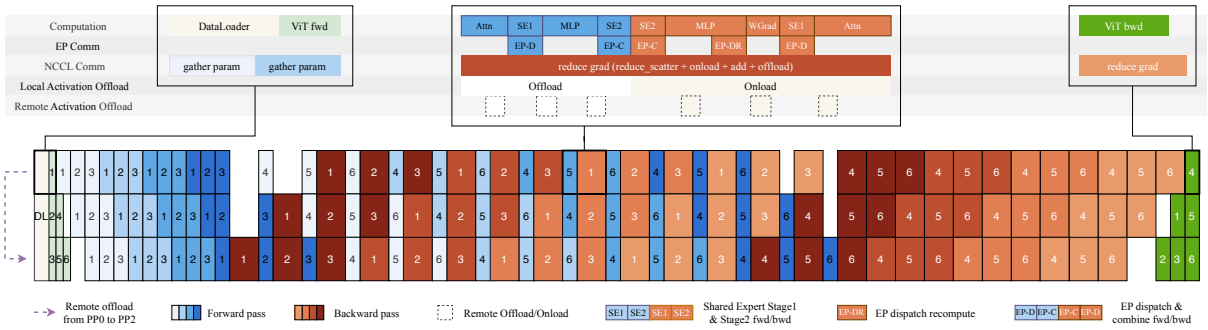


Figure 11: Computation, communication and offloading overlapped in different PP phases.

The MoE layers employ shared experts replicated across EP ranks, and the all-to-all communication for expert dispatch and combine is overlapped with computation to hide its latency.

Natively multimodal pre-training at the 3T-class poses three critical problems: (i) token loads are imbalanced across EP ranks; (ii) activations, gradients, and optimizer states exceed the memory budget; and (iii) the vision encoder's highly variable computation is exposed on the critical path. The following subsections address these problems in turn: perfectly balanced expert-parallel MoE training (§5.2.1), memory-efficient training (§5.2.2), and multimodal encoder optimization (§5.2.3). Fig. 11 illustrates the resulting execution schedule.

5.2.1 Perfectly Balanced Expert-Parallel MoE Training

In conventional EP schemes, token loads are imbalanced across ranks. The resulting computational imbalance degrades training throughput, and the dynamically varying shapes of routed-expert activations cause substantial memory fragmentation. We therefore propose MoonEP[3], an EP scheme that achieves perfect load balance with dynamic redundant experts. MoonEP preserves the overall computation flow of conventional schemes such as DeepEP [147] and additionally introduces online planning and migration of redundant experts. In the forward pass, we plan the redundant experts from the router outputs of the current micro-batch and layer and prefetch them before the routed-expert computation. In the backward pass, we stage their gradients in a local reduce buffer and, once the computation completes, reduce them back to the gradient buffers of their home ranks.

Perfect balance with bounded redundant experts MoonEP requires every rank to receive exactly $S \times K$ tokens, where S is the sequence length and K is the number of experts selected per token, so that all ranks perform identical amounts of computation. The key question is how many redundant experts suffice to guarantee such a balance. Let E be the number of experts and R the EP size. We prove that a balanced plan always exists with at most E/R redundant experts per rank and that this bound is essentially tight (§ E). Reserving E/R redundant-expert slots per rank therefore guarantees that planning always admits a feasible solution, so training is never interrupted. In contrast, prior work such as ECHO [137] and UltraEP [132] presets the number of redundant experts or imposes a per-rank token cap. Training is then forced to stop whenever no feasible plan exists within the cap, and the cap itself requires manual tuning while still leaving residual imbalance.

Online planning Computing the exact optimum at every training step is prohibitively expensive. We therefore compute exact solutions offline with integer linear programming (ILP) for representative cases as references and design a GPU planning kernel that is near-optimal, incurs negligible overhead, and always respects the E/R upper bound.

Zero-copy communication Perfect balance also simplifies the communication path. We implement a fused permute/unpermute operator in which the planning kernel precomputes the destination of every token, so tokens are sent directly to their expert-grouped positions on remote ranks, and views of the communication buffer are returned directly to the computation, eliminating intermediate copies. Under worst-case imbalance, supporting the same copy-free data path in DeepEP requires a communication buffer of size $S \times K \times R$, whereas MoonEP requires only a fixed $S \times K$ buffer owing to the perfect balance.

Sync-free execution with static shapes In conventional MoE implementations, the per-expert token counts vary across steps and layers, and the host must synchronize with the device at every layer to obtain the actual computation

[3] https://github.com/MoonshotAI/MoonEP

shapes before launching the expert computation, stalling the pipeline between layers. With perfect balance, every rank receives exactly $S \times K$ tokens and the computation shapes of all layers are statically known. This eliminates the per-layer MoE host synchronization and alleviates the host-side kernel-launch overhead.

Expert-GEMM scheduling and overlap Even with the aggregate load perfectly balanced across ranks, the per-expert token counts within each rank remain skewed, and a fixed-order, workload-oblivious schedule turns this skew into an imbalanced makespan across SM workers. We therefore schedule the routed-expert GEMM with a workload-aware scheduler that adapts its parameters to the current token distribution before launch and keeps them fixed during execution. A lightweight heuristic selects these parameters using an analytical cost model of hardware metrics, with key coefficients calibrated through offline autotuning. For the shared experts, we dispatch their GEMMs to a separate stream so that they overlap with other kernels.

5.2.2 Memory-Efficient Training

Unified activation manager We design a unified storage abstraction for activations, in which every tensor saved for the backward pass is associated with a pluggable storage backend. Recomputation, quantization, and offload/remote-offload are merely storage policies under this abstraction and can be freely composed at tensor granularity; policies are declared via lightweight annotations on tensors, fully decoupled from the model code. Recomputation is performed at function granularity, which supports cross-layer recomputation. In our implementation, all GPU memory is allocated on the main compute stream and managed within a single memory pool, avoiding multi-stream fragmentation and host-bound overhead; activations are prefetched back at layer granularity and overlapped with computation, introducing negligible extra overhead. In Kimi K3, most activations use block-wise FP8 quantization [58, 30] combined with offload/remote-offload, and element-wise operators are configured with recomputation.

Memory-efficient MoE In the native MoE implementation, the gradient computation of permuted probs depends on the forward output `output`. Inspired by SonicMoE [41], we rewrite this gradient through a mathematical transformation into a form that depends only on the intermediate activation `act_output` and the upstream gradient `doutput`, eliminating the backward dependency on `output` at the cost of an additional lightweight element-wise computation. Furthermore, in the forward pass of the group GEMM, we save only the input of the dispatch operation; during the backward pass, the input of the group GEMM is recovered by recomputing dispatch. As shown in Fig. 11, the communication introduced by this recomputation can be overlapped with part of the group-GEMM backward computation, eliminating this portion of activation storage at a negligible cost.

Memory-efficient Attention residual For the attention residual, we design a companion optimization based on Block AttnRes. The block representation is generated once at the boundary layer and shared by all subsequent layers, residing directly on the GPU. The AttnRes computation is entirely wrapped with checkpointing, so the activation saved for the backward pass at each layer is identical to that of the standard residual architecture. For pipeline parallelism, we adopt cache-based pipeline communication [57], in which only newly generated blocks are incrementally transferred between stages and released as soon as the micro-batch finishes, reaching the theoretical lower bound on memory footprint.

Balancing activations across PP ranks Under interleaved 1F1B pipeline parallelism, activations are unevenly distributed across PP ranks due to pipeline warmup, and the number of resident activations decreases as the PP rank increases. To avoid out-of-memory (OOM) errors, we remotely offload activations to the memory of other PP ranks using the Mooncake Transfer Engine [96], achieving balanced activation memory across PP ranks.

Pipeline ZeRO-2 gradient sharding and offloading Beyond activations, we use Pipeline ZeRO-2 gradient sharding [145] to shard gradients across data-parallel (DP) ranks. Furthermore, we store the sharded gradients in CPU memory to reduce peak GPU memory usage, while keeping the double grad buffer on the GPU. After gradients are reduced across DP ranks into the double grad buffer, they are accumulated into the CPU shards.

P2P-based Muon orthogonalization The distributed optimizer shards parameters evenly across DP ranks, whereas the Newton–Schulz orthogonalization in Muon requires the full parameter matrix, necessitating a communication step to gather complete parameters before each update. The naive approach performs an all-gather over the entire parameter buffer on every rank [73], which incurs a substantial memory footprint on top of making communication the primary bottleneck at scale. Instead, each rank retrieves only the shards of its locally owned parameters via peer-to-peer (P2P) communication with the corresponding owner ranks, eliminating the full-parameter buffer and reducing both memory usage and communication volume. Communication and computation are further pipelined at the granularity of model-chunk buffers, hiding the communication overhead.

5.2.3 Multimodal Encoder Optimization

Dynamic CP in multimodal encoder In long-context multimodal training, large images and long videos substantially increase the computation time of the vision encoder and cause significant load imbalance across devices. To address this, we extend context parallelism to such large samples. A single large image is partitioned along the patch dimension across multiple devices, and attention is computed by gathering key–value pairs (gather-KV) across CP ranks. In addition, we divide each CP group into several sub-CP groups and distribute multiple large images across them in a load-balanced manner, preventing the communication fraction from growing with scale. This reduces both the encoder latency of large visual samples and the cross-device load imbalance, allowing the remaining encoder computation to be hidden in pipeline bubbles.

Encoder computation in PP bubbles In Kimi K2.5, we introduced the Decoupled Encoder Process (DEP) [59], which splits ViT and text training into separate stages and balances vision forward and backward passes across PP stages. We observe that, under the interleaved 1F1B pipeline schedule, the text forward passes of the first PP micro-batches are all scheduled at the very beginning, while the text backward passes of the last PP micro-batches finish only at the very end. We therefore further decompose the ViT computation. The ViT forward passes of the first PP micro-batches are executed synchronously upfront, the remaining forward passes are scheduled into pipeline bubbles, and the backward passes are handled analogously. As a result, most of the ViT computation is hidden within pipeline bubbles, largely eliminating the effective overhead of the vision encoder.

5.3 Infra for 1M Agentic RL

Scaling agentic RL for a model as large as Kimi K3 to million-token contexts under a bounded compute budget makes resource efficiency a first-order goal. This motivates two complementary efforts: 1) efficient training and rollout, including KV-cache management, request scheduling, and training-state placement; 2) high-performance resumable sandboxes for long-horizon interaction.

5.3.1 Long-context RL infrastructure

We adopt co-located RL training [58] to keep each 1M-context Kimi K3 RL experiment within a few hundred GPUs, and use partial rollouts [118] to reduce tail latency from ultra-long trajectories. This design achieves good hardware utilization, but introduces a memory usage contention between rollout KV-cache that needs to be persisted for the next iteration, and the memory needed for training. This challenge becomes more severe in long-context RL.

External KV cache pool At 1M-context multi-step rollout, a prefix KV-cache miss is extremely expensive. Partial rollout exacerbates this issue at the beginning of each iteration, due to many unfinished long prefill requests from the previous iteration arriving at the same time. Speculative decoding further accelerates request turnover within relatively fixed tool-call intervals, increasing prefix-block churn. These issues can trigger preemption and lower the cache hit rate, which is critical for long-context RL.

We therefore decouple prefix retention from GPU residency with a write-back design. Active decoding blocks remain in GPU KV cache, while reusable idle prefixes are written back to an *external KV cache pool* in CPU DRAM only when it is evicted from GPU, and is prefetched back before the next reuse. KDA states are offloaded and prefetched together with the corresponding MLA KV cache blocks, keeping their lifecycles aligned. Compared with a write-through strategy, this policy incurs CPU DRAM usage and transfer bandwidth only for prefixes that leave the active decode path, avoiding redundant CPU copies of blocks that are still resident and active on GPU.

To provide sufficient DRAM for the external pool, we offload training states (model weights and optimizer states) to NVMe after a training iteration finishes. After a rollout iteration, the pool is released to avoid contention with training workloads.

Rollout auto-throttling scheduler In multi-step rollout, contexts grow progressively as the trajectory advances, making fixed concurrency based on the full-trajectory average length both hard to estimate and overly conservative early on. Conversely, setting concurrency too high creates KV cache pressure in later stages and can trigger preemption. We therefore design an auto-throttling mechanism at the LLM request scheduling layer, using runtime signals such as active request count, queued request count, and KV cache utilization to dynamically control how many requests are sent to the inference engine. This keeps early rollout well utilized while reducing concurrency as KV cache pressure rises, avoiding both under-saturation and overload without manual tuning.

Gradient-buffer reuse for non-policy model forwarding RL loss computation often requires forward-only non-policy models, such as reference models, whose weights are too large to keep resident on GPU. We keep these weights in CPU memory and materialize them only when needed, backing their parameter tensors with the policy model's FP32 gradient-buffer storage. This reuses existing GPU memory without extra allocation or fragmentation, and remains safe because the buffers are overwritten when real gradients are later computed.

With ZeRO-2 gradient sharding and offloading (§ 5.2.2), each GPU retains gradient buffers for only two VPP chunks in Kimi K3 RL training. We stream reference weights into these slots chunk by chunk: one slot is used for the current forward computation while the other prefetches the next chunk, hiding copy overhead without increasing GPU memory.

5.3.2 Sandbox Infrastructure

We employ multiple sandbox runtimes to support the diverse requirements of Kimi K3 post-training and evaluation, including a traditional container-based runtime, a GPU sandbox runtime, and, most notably, a new microVM-based sandbox runtime called AgentENV.

AgentENV[4], developed in collaboration with our partners, is a sandbox system specifically designed for agentic AI workloads. It is built around three core design goals:

- **High-fidelity isolated sandbox runtime** As agents become more capable and tasks more difficult, they tend to explore more aggressively and may even attempt reward hacking. On the one hand, this poses unique security challenges: in our early experiments with traditional container-based sandbox runtimes, we observed several kernel panics and deadlocks caused by unintended agent operations. On the other hand, we want to permit as much exploration as possible so as not to constrain agent capability, and complex tasks require a sandbox close to a real-world environment — for example, agents should be able to mount disks, run containers, or even launch virtual machines at will. By running isolated microVMs with Firecracker [3], AgentENV provides a level of isolation and fidelity that container-based runtimes cannot match.
- **Flexible sandbox life-cycles for agentic RL** At the low level, AgentENV supports incremental checkpointing and resuming of sandbox states, where only memory pages dirtied since the last checkpoint are saved during checkpointing, achieving checkpoint and resume latencies as low as 133 ms and 49 ms, respectively. On top of this, AgentENV provides three high-level operations that help improve agentic RL efficiency. **(a) Pause and Resume**: a paused sandbox consumes no memory or CPU resources; a sandbox can therefore be paused while the agent is waiting for the model's inference result, which can account for as much as 98% of the sandbox lifetime. **(b) Fork**: fork creates a new sandbox from the exact state of the original one while keeping the original running, which is useful for reward judging without side effects. **(c) Snapshot**: snapshots of a sandbox can be saved at regular intervals for error recovery.
- **High efficiency and high density** In our workloads, tens of thousands of sandboxes, each with a unique set of images, may need to be created within seconds. We adopt OverlayBD [68] as the image format, together with a custom ublk driver implementation, storage-layer sharing, and P2P transport, achieving sub-second launch latency at large scale. We further reduce memory usage with copy-on-write memory and page-cache optimizations, achieving a memory overcommit ratio of up to 6.5× in real workloads.

Throughout Kimi K3's training and evaluation, a total of 51,219,741 sandboxes across 1,505,678 images were created.

5.4 Inference and Online Serving

Serving Kimi K3 exposes the same challenges from the production side: the hybrid KDA–MLA architecture maintains two fundamentally different caches that must be managed jointly at million-token contexts, its new modules and highly sparse experts demand kernels tailored to each, and production traffic mixes requests whose per-request cost spans three orders of magnitude. The designs below address these challenges at three levels. At the engine level, a KDA-aware prefix cache packs the fixed-size recurrent state into the same paged pool as the MLA KV cache and keeps long prefixes reusable across requests. At the device level, dedicated kernels for KDA decoding, Block AttnRes, and the sparse latent MoE minimize per-token latency and memory traffic. At the fleet level, cache-aware affinity scheduling and budget-based admission control translate these efficiencies into predictable serving.

5.4.1 KDA-Aware Prefix Cache Management

The hybrid architecture in Kimi K3 complicates prefix caching: the KDA recurrent state and the MLA KV cache differ fundamentally in size and lifetime, yet a cached prefix is reusable only when both can be restored together at the same

[4]AgentENV is open-sourced at https://github.com/kvcache-ai/AgentENV

boundary. We therefore design a KDA-aware prefix cache that manages the two cache types jointly—from a unified paged layout to fine-grained prefix reuse and consistency under concurrent scheduling—keeping million-token prefixes cheap to retain and reusable across requests.

Unified cache layout for hybrid KDA–MLA attention Each Kimi K3 block consists of three KDA layers and one Gated MLA layer, whose caches differ fundamentally. The MLA KV cache grows with sequence length and is paged per token, whereas the KDA recurrent state is fixed in size with a single copy per request. Maintaining a separate manager for each would duplicate the allocation, eviction, and transfer logic. We therefore pack KDA states into the same paged block pool as MLA KV, unifying pages to the same byte size so that both page types share one implementation of allocation, reference counting, and eviction. Within a page, the states of all heads are stored contiguously head by head, so that each head's byte stream is self-contained and serves as the minimal unit of cross-node transfer. Under prefill/decode disaggregation, when prefill and decode nodes adopt different TP degrees, re-layout is performed on the transfer path with zero GPU-side reshuffling. This asymmetry proved useful during development: any type-confused access yields garbage rather than plausible data — a zero-overhead sanity check on the pooled layout.

KDA prefix cache optimization Block-hash-based prefix caching reuses the KV cache at the granularity of one physical block: only complete blocks are hashed, so only block-aligned prefixes are reusable.

This coupling breaks down in Kimi K3. Block-hash matching requires one block size shared by all layers, and a prefix hit is reusable only if the KDA state at the hit boundary has been persisted. A KDA layer maintains a single large recurrent state per sequence rather than per-token entries, so state snapshots are affordable only at sparse boundaries; the shared block size is therefore forced to 1024–6144 tokens—and, since hashing is tied to the storage block, the hash granularity as well, although MLA's per-token entries alone would tolerate much finer blocks. At such a coarse granularity caching is nearly useless: requests shorter than one block can never be reused, and chunked prefill exports no cacheable prefix until it crosses a full block boundary.

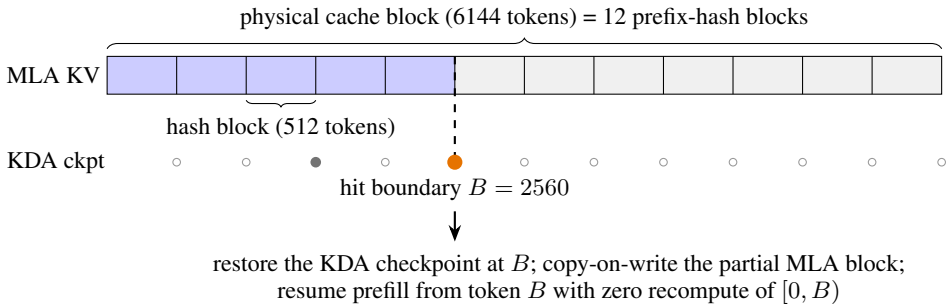


Figure 12: **Fine-grained prefix caching within a physical cache block.** A 6144-token physical block contains twelve 512-token hash blocks, with cached MLA blocks shown in blue and empty blocks in light gray. The markers below show the KDA checkpoint status at each hash boundary. An open circle (○) denotes a boundary without a stored checkpoint, a gray dot (●) denotes a persisted KDA checkpoint, and an orange dot (●) marks the checkpoint hit at $B = 2560$. Persisted checkpoints are sparse and typically coincide with conversation-turn boundaries. The request reuses the five MLA hash blocks and the KDA checkpoint at B, then resumes prefill without recomputing $[0, B)$.

We therefore decouple the two granularities. Prefix hashing runs on fine *hash blocks* (e.g., 512 tokens) inside MLA pages, while the physical block remains the coarse allocation unit. Alignment runs the other way for KDA: checkpoints of the recurrent state are saved only at (a sparse subset of) MLA's hash endpoints—the only positions a lookup can ever reference.

During prefill, a partially filled MLA page is registered in the prefix-cache index under the chained hash of its last complete hash block, where each hash covers all preceding hash blocks so that matching an endpoint certifies the whole prefix up to it; the registered endpoint advances as the page fills. Meanwhile, after each forward pass, the KDA kernel persists the recurrent state at the last hash-aligned position processed. Checkpoints are large, so intermediate checkpoints superseded as the request advances are recycled, while those at conversation-turn boundaries are retained for cross-request reuse. Cached checkpoints are read-only snapshots: a hit restores the state by copying it into the request's private running state before the next forward pass, and new checkpoints are written to fresh slots, so a checkpoint visible to other requests is never mutated in place.

Lookup proceeds in two stages (Fig. 12). The MLA stage matches whole physical blocks by chained hash and, at the first missing block, falls back to the hash endpoints inside it, so partially filled pages remain hittable. The KDA stage then requires a checkpoint at the candidate boundary in every KDA cache group, each of which maintains an

independent recurrent state. The hit is the longest boundary satisfying both stages—always a multiple of the hash block, and never required to be a multiple of the physical block. In Fig. 12, a request whose first 2800 tokens match the cached prefix hits at $B = 2560 = 5 \times 512$, deep inside a 6144-token physical block, and resumes prefill from token B instead of recomputing $[0, B)$.

Consistency under concurrent scheduling The remaining design points are each dictated by a concrete failure mode of sharing partially filled blocks, in a setting where a hit block is at once a shared cache entry and the growth point of a private request, and where the MLA and KDA cache groups must agree on every hit boundary. First, all cache groups draw blocks from one shared free list, so allocating a private copy for one group could evict a block that another group has just hit; every hit block is therefore pinned across all groups before anything is allocated. Second, the copy into the private block executes on the GPU immediately before the forward pass, so a block allocated or registered within the current scheduling step would still hand the previous owner's bytes to a reader; such blocks are excluded from matching until their copies land. Third, a checkpoint can restore a request only if it exists in every KDA group, so evicting one group's checkpoint atomically invalidates its siblings — a checkpoint is either hittable in every group or in none. With these mechanisms, every registered state always corresponds to exactly its declared token prefix, and prefix caching for hybrid KDA–MLA models reaches the same generality as for full-attention models: any shared prefix is reusable at any 512-token boundary, independently of request length, chunking, or scheduling interleaving.

5.4.2 High-Performance Kernels

Kimi K3 introduces several new architectural modules: KDA (§2.1.1), Block AttnRes (§2.2), and Stable LatentMoE (§2.3). We optimize the kernel implementation for each.

KDA Compared with KDA prefill (§5.1), KDA decoding presents a distinct set of challenges: the primary bottleneck shifts from exploiting parallelism to efficiently managing the evolving recurrent state, which is updated in place at every decoding step. This in-place update becomes problematic in MTP-based speculative decoding: if verification rejects a subset of the drafted tokens, the state has already advanced beyond the last accepted token and cannot be trivially rolled back. Maintaining a state snapshot for each draft position would enable rollback, but would also multiply state traffic — a cost that dominates at the large batch sizes typical of online serving.

The state after any accepted draft prefix, however, is fully determined by the projected inputs of the draft tokens, which are far smaller than the state itself. We therefore cache only these projected inputs, rebuild the states of accepted tokens on-chip, and write back the states of the verified and bonus tokens, a design independently proposed in the concurrent work ReplaySSM [25]. The replayed tokens, the bonus token, and the next draft window share one recurrent loop inside a single fused kernel covering short convolution, input normalization, gating, the KDA recurrence, and output normalization. Verification latency grows sub-linearly with the number of tokens verified and remains below that of state-caching baselines. Because the projection caches never leave the decode stage, prefix caching and prefill–decode disaggregation operate on the same payload as in non-speculative serving.

Block AttnRes Block AttnRes [57] follows a two-phase schedule: a batched inter-block pass reads the cached block representations once per block, after which each layer folds in the intra-block partial sum through an online-softmax merge [79]. Memory access accounts for a substantial fraction of the cost of these kernels in both prefill and decoding, so our optimizations in both stages focus primarily on memory efficiency.

For prefill, materializing the block representations on every tensor-parallel (TP) rank would incur substantial redundant memory consumption. We therefore adopt sequence parallelism (SP) for activations: the TP all-reduce is decomposed into a reduce-scatter and an all-gather, with the intra-block kernel inserted between the two collectives, operating on the sequence-sharded hidden states so that the block representations of each token are materialized on exactly one rank. This eliminates the additional memory consumption and reduces the I/O overheads of Block AttnRes during prefill.

For decoding, we launch the inter-block kernel on a side stream so that it overlaps with independent computation on the main stream. The intra-block kernel is instead streamlined through fusion: the merging of the AttnRes output with its partial-sum update, together with the subsequent RMSNorm, is fused into the preceding TP all-reduce, eliminating a dedicated kernel for the intra-block phase. Together, these optimizations hide the latency of the inter-block pass and reduce the memory traffic of the intra-block phase.

Stable LatentMoE Stable LatentMoE increases both the total number of experts and the number of activated experts per token. The resulting growth in both the expert space and the per-token expert count raises scheduling and coordination overheads, making it difficult for conventional MoE kernels to sustain high hardware utilization. These challenges motivate dedicated kernel optimizations for this module.

To mitigate the overhead of the latent GEMMs, we adopt three optimizations. First, we fuse the latent down-projection with the MoE router into a single GEMM. Second, we shard latent weight matrices across ranks and fuse the output all-gather into the GEMM epilogue using multimem store instructions. Finally, we overlap the resulting communication with other operators, such as the shared-expert computation. Together, these optimizations eliminate redundant weight traffic and duplicated computation, while hiding the communication latency behind computation.

For routed experts, at small batch sizes, the group GEMMs reduce to memory-bound streaming of weight matrices — a regime for which conventional tile-centric kernels are poorly suited due to their compute-oriented design and preprocessing overheads. We instead build the MoE decoding kernel upon the token-centric design of WarpDecode [12], in which each warp is responsible for one output neuron and streams the associated weights directly from memory. To further increase parallelism, we subdivide each warp into finer-grained lane teams, each processing a disjoint subset of experts, followed by a warp-wide reduction of the partial results. In addition, the weight layout is permuted offline at a one-time preprocessing cost, substantially reducing the runtime dequantization overhead.

5.4.3 Fleet-Level Scheduling

Beyond a single serving instance, the challenge shifts from per-request efficiency to predictability: a prefix-cache miss costs orders of magnitude more than a hit, and a burst of million-token requests can starve short ones. We propose two fleet-level scheduling policies to address this: cache-aware affinity scheduling routes each session to the cluster holding its prefix cache while bounding the cost of cluster failures, and budget-based admission control grants each request class its own resource budget so that bursty long-context traffic cannot degrade system-wide SLOs.

Cache-aware affinity scheduling At 1M context, a typical coding input carries a prefix of 400K tokens but requires a prefill increment of only 4K tokens, so a prefix-cache hit avoids re-prefilling the entire prefix and is orders of magnitude cheaper than a miss. We therefore route each request to the cluster that holds its prefix cache, as moving the cache to another cluster would require transferring it over inter-cluster links far slower than the intra-cluster fabric. This cache-aware affinity, however, binds each session to a single cluster, whose failure would interrupt all sessions bound to it. Consistent hashing therefore pins each session to two clusters, a primary that serves its traffic and a pre-assigned secondary that takes over when the primary fails. The secondary holds none of the session's prefix cache and must re-prefill it upon failover. Since consistent hashing distributes the secondary assignments of different sessions uniformly across the fleet, this re-prefill work is divided among many clusters rather than concentrated on one. Cache locality is thus preserved in the common case, while the impact of any single cluster failure remains bounded.

Budget-based admission control Production traffic mixes short requests under 2K tokens with ultra-long requests up to 1M tokens, so the per-request cost spans roughly three orders of magnitude and the total load imposed by any fixed number of requests is highly unpredictable. Capacity planning, queueing models, and rate-limiting quotas based on the "average request" all break down under this variance. In a typical failure mode, a burst of long-context requests saturates the available compute, and short requests arriving afterwards cannot be scheduled promptly, degrading time to first token (TTFT) across all traffic. We therefore adopt budget-based admission control, allocating separate resource budgets to different request classes so that bursty long-context traffic consumes at most its own share of the capacity and cannot degrade system-wide SLOs experienced by other classes.

6 Evaluations

6.1 Main Results

6.1.1 Benchmarks

We evaluate Kimi K3 on a comprehensive benchmark suite organized along four broad capability axes:

- **Reasoning & Knowledge**: GPQA Diamond [101], CritPt [8], AA-LCR [9], and Humanity's Last Exam (HLE-Full, with and without tools) [93].
- **Coding**: DeepSWE [31], ProgramBench [95], Terminal-Bench 2.1 [78], FrontierSWE [35], SWE-Marathon [117], PostTrainBench [94], MLS-Bench-Lite [76], and SciCode [121, 8].
- **Agentic**: BrowseComp [131], DeepSearchQA [126], ResearchRubrics [106], Toolathlon-Verified [69], MCPMark-Verified [133], MCP-Atlas [11], AutomationBench [108], JobBench [70], GDPval-AA v2 [90], AA-Briefcase [8, 2], Agents' Last Exam (ALE) [4, 115], APEX-Agents [127], OfficeQA Pro [87], SpreadsheetBench 2 [150], OSWorld-Verified [136] and OSWorld 2.0 [143], SaaS-Bench [109], τ^3-Banking [1, 8], Harvey Lab-AA [8, 42], CorpFin v2 [21], Finance Agent v2 [34], and Legal Research Bench [65].

- **Vision**: WorldVQA [149], OmniDocBench [88], PerceptionBench [62], Video-MME [36], MMVU [148], and BabyVision [13] with Python tool. MMMU-Pro [144], CharXiv (RQ) [130], Math-Vision [128], and ZeroBench-main [102], each with and without Python tool augmentation.

6.1.2 Baselines

We benchmark against state-of-the-art proprietary and open-source models. For proprietary models, we compare against Claude Fable 5 [16], GPT-5.6 Sol [39], Claude Opus 4.8 [17], and GPT-5.5 [38]. The results of Claude Fable 5 include fallback behaviors and the results of GPT-5.6 Sol include potential cyberguards. For open-source models, we include GLM-5.2 [37]. All models are evaluated at maximum reasoning effort, except GPT-5.5, which uses the "xhigh" setting.

6.1.3 Evaluation Configurations

All Kimi K3 evaluations use reasoning effort `max` and temperature = 1.0. For single-step tasks, such as GPQA Diamond, HLE-Full, and vision benchmarks without tools, we set top-p = 0.95. For agentic tasks, we set top-p = 1.0. Generally, we recommend using top-p = 0.95 for reasoning and knowledge tasks, and top-p = 1.0 for coding and agentic scenarios.

Coding Each model is evaluated under one of three agentic harnesses: Kimi Code [56], Claude Code [15], or Codex [20]. On DeepSWE, we report results on the v1.1 tasks, with additional reference to the official leaderboard (Kimi K3 attains 67.3 with the mini-SWE-agent harness). On Terminal-Bench 2.1, we report the best score across harnesses for all models. Our SWE-Marathon evaluation is based on an H20-calibrated branch of the official tasks as of July 9, 2026, prior to the final v1.1 release, with Docker images, performance gates, and reference oracles for the GPU tasks recalibrated for H20 but the correctness and anti-cheat validators unchanged; Claude Fable 5 hits fallbacks on 35% of the tasks. For PostTrainBench, we evaluate Kimi K3, Claude Fable 5, and GPT-5.6 Sol using the official Harbor implementation at maximum effort, averaged over three runs on H20 GPUs (instead of H100 in the official setting). FrontierSWE dominance scores are recomputed from raw scores using the official evaluation script as of July 16, 2026.

Agentic For OfficeQA Pro, each test case provides the agent with the entire PDF corpus rendered as images, with no machine-readable text available. MCP-Atlas is evaluated on the 500-task public subset with a 100-turn limit, using Gemini 3.1 Pro as the judge. AutomationBench is evaluated on the 600-task public subset. For BrowseComp we adopt a context-compaction strategy triggered at 300K tokens; evaluated with the full 1M-token context window and no context management, Kimi K3 achieves 90.4%.

Vision Scores are averaged over three runs, except ZeroBench-main, which we run five times following the official setting. MMMU-Pro follows the official protocol, preserving the original input order and prepending images to the text input. For WorldVQA, we observe consistent refusal behavior across models and enforce an answer via prompt engineering.

Third-party results GDPval-AA v2, AA-Briefcase, τ^3-Banking, Harvey Lab-AA, APEX-Agents, SciCode, AA-LCR, and CritPt scores are cited from Artificial Analysis [8] as of July 23, 2026. For Harvey Lab-AA, we report the criterion pass rate. CorpFin v2, Finance Agent v2, and Legal Research Bench scores are cited from Vals AI [124]. Agents' Last Exam scores are cited from the official leaderboard [4] as of July 23, 2026; we report the leaderboard's primary pass-rate metric. On the leaderboard, each model is paired with a specific harness: Kimi K3 with Kimi Code; GPT-5.6 Sol, GPT-5.5 with Codex; and Claude Fable 5, Claude Opus 4.8, and GLM-5.2 with Claude Code. Toolathlon-verified and JobBench scores are cited from their official leaderboards [119, 52] as of July 24, 2026.

6.1.4 Results

Table 2 provides a comprehensive comparison of Kimi K3 against both proprietary and open-source baselines. Overall, Kimi K3 closely trails the strongest proprietary models, Claude Fable 5 and GPT-5.6 Sol, while consistently outperforming Claude Opus 4.8, GPT-5.5, and GLM-5.2 across the benchmark suite. We highlight key observations across core capability domains below:

Reasoning & Knowledge On graduate-level reasoning, Kimi K3 is competitive with the frontier, scoring 93.5% on GPQA Diamond. However, a gap remains on research-level tasks: on HLE-Full it trails Claude Fable 5 and GPT-5.6 Sol both with and without tools, at 56.0% and 43.5% respectively; and on CritPt it scores 23.4%, lagging behind Claude Fable 5, GPT-5.6 Sol, and GPT-5.5, indicating that research-level reasoning remains a key direction for improvement.

Table 2: Performance comparison of Kimi K3 against proprietary and open-source models. **Bold** denotes the best result for each benchmark and underline the second-best. Unless otherwise noted, Kimi K3 results are obtained with reasoning effort set to `max` and temperature equal to 1.0. For HLE-Full, MMMU-Pro, CharXiv (RQ), Math-Vision, and ZeroBench, each cell reports the scores without and with tool augmentation (general tools for HLE-Full, Python for the vision benchmarks), in that order. †On the official Agents' Last Exam leaderboard, the Claude Fable 5 entry runs at xhigh effort with 40% of tasks annotated as downgraded.

		Proprietary				Open Weight
Benchmark	**Kimi K3 (max)**	**Claude Fable 5 (max, w/ fallback)**	**GPT-5.6 Sol (max)**	**Claude Opus 4.8 (max)**	**GPT-5.5 (xhigh)**	**GLM-5.2 (max)**
Reasoning & Knowledge						
GPQA Diamond	93.5	92.6	**94.1**	91.0	93.5	91.2
CritPt	23.4	28.6	**32.3**	20.9	27.1	20.9
AA-LCR	**74.7**	70.0	73.7	67.7	74.3	71.3
HLE-Full	43.5 / 56.0	**53.3 / 63.0**	44.5 / 58.0	49.8 / 57.9	41.4 / 52.2	-
Coding						
DeepSWE	67.5	70.0	**73.0**	59.0	67.0	46.2
ProgramBench	**77.8**	76.8	77.6	71.9	70.8	63.7
Terminal-Bench 2.1	88.3	88.0	**88.8**	84.6	83.4	82.7
FrontierSWE	81.2	**86.6**	71.3	66.7	64.9	67.3
SWE-Marathon	**42.0**	35.0	39.0	40.0	14.0	13.0
PostTrainBench	36.6	**41.4**	34.6	34.1	28.4	34.3
MLS-Bench-Lite	48.3	**49.9**	46.2	42.8	35.5	40.4
SciCode	58.7	**60.2**	56.1	53.5	56.1	50.5
Agentic						
BrowseComp	**91.2**	88.0	90.4	84.3	84.4	-
DeepSearchQA (F1)	**95.0**	94.2	-	93.1	-	-
ResearchRubrics	**76.2**	-	73.8	73.5	64.0	71.1
GDPval-AA v2 (Elo)	1686	**1747**	1736	1593	1491	1510
Toolathlon-Verified	76.5	**77.9**	74.9	76.2	73.5	59.9
MCPMark-Verified	**94.5**	87.4	92.9	76.4	92.9	-
MCP-Atlas	84.2	**84.7**	83.6	83.6	82.8	82.6
AutomationBench	**30.8**	29.1	29.7	27.2	22.7	12.9
JobBench	54.3	**57.4**	45.4	48.4	38.3	43.4
AA-Briefcase (Elo)	1548	**1583**	1495	1354	1158	1260
Agents' Last Exam	28.3	25.7†	**29.6**	27.0	26.6	20.4
APEX-Agents	41.0	**43.3**	39.9	39.4	38.5	35.6
OfficeQA Pro	63.3	**69.9**	63.2	63.9	60.9	41.4
SpreadsheetBench 2	**34.8**	34.7	32.4	31.6	29.1	28.1
OSWorld-Verified	84.8	**85.0**	83.0	83.4	79.0	-
OSWorld 2.0	58.3	**66.1**	62.6	55.7	49.5	-
SaaS-Bench	60.1	-	**61.4**	56.1	43.8	-
τ^3-Banking	**33.4**	26.8	33.0	27.6	31.3	26.8
Harvey Lab-AA	**94.6**	93.6	87.2	91.1	86.3	91.0
CorpFin v2	71.6	**71.8**	64.4	66.7	68.4	66.1
Finance Agent v2	54.4	**56.3**	53.8	53.9	51.8	49.7
Legal Research Bench	44.2	**49.5**	48.1	43.8	40.4	31.3
Vision						
WorldVQA ForceAnswer	51.0	**56.7**	41.8	39.1	38.5	-
OmniDocBench	**91.1**	89.8	85.8	87.9	89.4	-
PerceptionBench	58.5	57.2	**59.7**	47.2	55.8	-
Video-MME (w/ sub)	**90.0**	-	89.5	86.0	89.3	-
MMVU	**82.1**	-	81.2	79.2	81.7	-
BabyVision w/ Python	85.7	**90.5**	88.9	81.2	83.6	-
MMMU-Pro	81.6 / 83.4	81.2 / **86.5**	**83.0** / 84.6	78.9 / 82.7	81.2 / 83.2	-
CharXiv (RQ)	84.8 / 91.3	**88.9 / 93.5**	84.6 / 89.1	80.5 / 89.9	84.1 / 89.0	-
Math-Vision	94.3 / 97.8	94.8 / **98.6**	**95.8** / 97.8	86.7 / 97.1	92.2 / 96.8	-
ZeroBench-main (pass@5)	**23.0** / 41.0	**23.0 / 46.0**	17.0 / 35.0	17.0 / 34.0	22.0 / 41.0	-

Coding Kimi K3 delivers strong agentic coding performance. It attains the best score on ProgramBench (77.8%), and on SWE-Marathon—a GPU-kernel-oriented suite—it scores 42.0%, 7 points ahead of Claude Fable 5. On Terminal-Bench 2.1, it nearly matches GPT-5.6 Sol (88.3% vs. 88.8%). On DeepSWE, it ranks behind Claude Fable 5 and GPT-5.6 Sol but ahead of Claude Opus 4.8 and GPT-5.5. On FrontierSWE, a long-horizon benchmark, it ranks second with a score of 81.2% as of July 16, 2026, behind only Claude Fable 5 (86.6%) and well ahead of all other models.

Agentic Kimi K3 achieves state-of-the-art results on a broad set of agentic suites, including BrowseComp (91.2%), DeepSearchQA (95.0% F1 score), ResearchRubrics (76.2%), MCPMark-Verified (94.5%), AutomationBench (30.8%), SpreadsheetBench 2 (34.8%), τ^3-Banking (33.4%), and Harvey Lab-AA (94.6% criterion pass rate). The main exceptions are the Elo-rated knowledge-work suites, both led by Claude Fable 5: Kimi K3 places third on GDPval-AA v2 (1,686) and second on AA-Briefcase (1,548). Elsewhere it is largely competitive: on CorpFin v2 and OSWorld-Verified, it finishes just 0.2 points behind Claude Fable 5 (71.6% vs. 71.8% and 84.8% vs. 85.0%, respectively), while the remaining harder computer-use benchmarks (OSWorld 2.0, SaaS-Bench) are still led by Claude Fable 5 or GPT-5.6 Sol.

Vision Kimi K3 exhibits strong multimodal understanding capabilities, which are further amplified by Python tools: on Math-Vision it reaches 94.3%, rising to 97.8% with Python tools, and on the challenging ZeroBench-main it ties Claude Fable 5 at 23.0% (pass@5), jumping to 41.0% with Python tools. It also achieves the highest score on OmniDocBench (91.1%) and, on WorldVQA (51.0%), ranks second behind Claude Fable 5, ahead of GPT-5.6 Sol and Claude Opus 4.8.

6.2 Internal Evaluation

6.2.1 Capability Evaluation

Beyond the public benchmark suite, we maintain a collection of in-house benchmarks that target capability areas public evaluations do not adequately cover, giving a more comprehensive measure of model and agent capabilities. These benchmarks are refreshed and expanded frequently, so that they can closely track the model's evolving failure modes and directly guide data and training iterations. They broadly fall into three categories: coding capability and experience, general agent experience, and conversational experience. Table 3 reports the results across these benchmarks.

Coding Capability and Experience

- **Kimi Code Bench 2.0 (KCB 2.0)**: evaluates code agents on realistic, end-to-end software engineering tasks across a broad range of programming languages and production-oriented technology stacks.
- **Kimi Webdev Bench**: evaluates models on challenging web development prompts drawn from real usage scenarios, with outputs compared through blind expert judgment, with results available in Table 4.
- **Coding Experience**: evaluates the practical experience of working with the model as a coding agent in real development workflows.

General Agent Experience

- **24/7 ClawBench 2.0**: simulates always-on assistant work, in which tasks span multiple days, events arrive concurrently, and interruptions are routine.
- **Multi-Agent Infra for Routing and Assignment (MIRA) Bench**: evaluates long-chain, multi-role, multi-system enterprise collaboration tasks, assessing whether agents can carry out end-to-end work and judge when to organize or delegate to subagents.
- **Kimi Autonomous Execution Tasks (KAET)**: evaluates long-horizon autonomous execution on tasks simulating real user requests and enterprise system operations.
- **Context Learning and Instruction Following (CLIF) Bench**: targets in-context learning, requiring models to learn from a provided context while following instructions that interleave multiple complex skills.
- **Agentic Vision Bench**: evaluates whether agents notice and correctly use key visual facts during task execution.
- **Swarm Bench**: evaluates models' ability to orchestrate agent swarms [59] on complex tasks that benefit from coordinated decomposition and parallel execution.
- **Online Experience**: mirrors the distribution of real online agent usage, measuring performance on the deliverable file types most frequently requested by users.

Table 3: Results on our in-house benchmarks. **Bold** denotes the best reported result per benchmark; "-" denotes scores not yet included in this report. Unless otherwise noted, models are evaluated at maximum reasoning effort (GPT-5.5 at xhigh); harness assignments are shown in the Harness column. [a]13 fallbacks and 1 refusal out of 80 tasks. [b]10 refusals out of 80 tasks. [c]3 refusals out of 80 tasks. [d]Includes 2 tasks that Claude Fable 5 refused to answer. [e]Includes 14 tasks that Claude Fable 5 refused to answer. [f]6 refusals out of 95 tasks. [g] Reported metric is 1−hallucination rate; higher is better.

			Proprietary				Open Weight
Benchmark	**Harness**	**Kimi K3 (max)**	**Claude Fable 5 (max)**	**GPT-5.6 Sol (max)**	**Claude Opus 4.8 (max)**	**GPT-5.5 (xhigh)**	**GLM-5.2 (max)**
Coding Experience							
Kimi Code Bench 2.0	Claude Code	73.7	**76.9**[a]	-	71.7	-	64.2
	Kimi Code	72.9	-	-	-	66.0	-
	Codex	-	-	64.8[b]	-	69.0[c]	-
Coding Experience	Claude Code	**59.9**	59.8	-	58.0	-	53.3
	Kimi Code	56.6	-	-	-	-	-
	Codex	-	-	59.3	-	56.8	-
General Agent Experience							
24/7 ClawBench 2.0	OpenClaw	48.3	47.4[d]	**52.0**	47.2	48.5	43.2
MIRA Bench	MIRA	64.1	**72.9**	62.2	59.8	54.6	-
KAET	Kimi Code	83.5	-	**85.4**	78.7	79.7	74.7
CLIF Bench	Kimi Code	**52.4**	-	50.6	48.8	52.3	39.2
Agentic Vision Bench	Kimi Code	78.3	81.1	**82.9**	82.8	76.9	-
Swarm Bench	Kimi Agent	**76.3**	-	73.2	72.6	61.8	58.5
Online Experience	Kimi Agent	77.9	74.2[e]	**84.0**	69.4	73.7	64.0
Deep Research Bench	Kimi Agent	**90.0**	-	85.3	87.2	81.9	84.0
Finance Bench	N/A	62.6	-	**62.7**	60.7	58.4	55.4
KWV Bench	N/A	64.7	63.6	**66.9**	61.7	65.8	-
DECK Bench	N/A	73.5	73.0	**74.7**	66.9	68.2	68.6
Agent Behavior Bench	Kimi Work	65.0	75.5[f]	**76.4**	65.7	70.1	-
Conversational Experience							
Faithfulness [g]	N/A	85.5	-	84.8	83.6	**86.5**	74.8
Chat All-in-One Bench	Kimi Work	85.2	**88.0**	79.0	83.8	71.8	-

Table 4: Results on the in-house Kimi Webdev Bench: Kimi K3 (max) against Claude Opus 4.8 (max), both run with the Claude Code harness. The comparison is performed under blind expert judging, where experts score each output on code quality, feature completeness, visual fidelity, and interaction experience without knowing which model produced it. Win, Tie, and Lose report the percentage of prompts where Kimi K3's output is preferred, rated comparable, or dispreferred, respectively.

Domain	**Win**	**Tie**	**Lose**	**Win − Lose**
Games	55.6%	3.7%	40.7%	+14.9%
3D / WebGL / Shader	72.7%	13.7%	13.6%	+59.1%
Website / UI Clone	52.6%	21.1%	26.3%	+26.3%
Overall	58.6%	13.8%	27.6%	**+31.0%**

- **Deep Research Bench**: evaluates models on deep-research-style queries curated by domain experts and graded with expert-aligned rubrics.
- **Finance Bench**: evaluates models on realistic financial work that requires end-to-end execution of complete workflows, from source materials to reviewable deliverables.
- **Knowledge Work Vision (KWV) Bench**: evaluates atomic visual capabilities extracted from tasks distilled from real knowledge-work scenarios.
- **DECK Bench**: measures the capability to produce high-quality presentation decks from task descriptions drawn from real usage scenarios.
- **Agent Behavior Bench**: extends agent evaluation from outcome correctness to process quality, scoring tool-use behavior, efficiency, and discipline alongside task completion.

Conversational Experience

- **Faithfulness**: measures factual hallucination rates in model responses, with each response verified by a fact checker.
- **Chat All-in-One Bench**: measures conversational experience at every stage of product usage, with scenarios designed around real online user needs.

Evaluation Configurations Unless a benchmark is split into separate rows by harness, the Harness column in Table 3 reports the harness used for Kimi K3. For other models, Claude models and GLM-5.2 are evaluated with Claude Code, while GPT models are evaluated with Codex. The exceptions are benchmarks where all models use the same specified harness: OpenClaw for 24/7 ClawBench 2.0; MIRA (Multi-Agent Infra for Routing and Assignment), an internal out-of-distribution harness, for MIRA Bench; Kimi Work for Agent Behavior Bench and Chat All-in-One; and Kimi Code for CLIF and Agentic Vision Bench.

Results The in-house suite separates Kimi K3's strengths from its weaknesses more sharply than the public benchmarks. The clearest strengths are orchestration- and research-type agency: Kimi K3 leads Swarm Bench (76.3) and Deep Research Bench (90.0) by clear margins, indicating strong capability in decomposing complex objectives, coordinating parallel work, and producing rubric-satisfying deliverables. Coding is likewise a strength: on Kimi Code Bench 2.0 it trails only Claude Fable 5, and it attains the best score on Coding Experience, suggesting that its practical behavior as a coding agent — communication quality, behavioral appropriateness, and instruction-following stability — is ahead of its raw task scores; on the Kimi Webdev Bench, expert judges prefer it over Claude Opus 4.8 by a +31.0-point overall margin, with the largest gain on 3D/WebGL/Shader tasks. Professional knowledge work has also improved markedly over the previous generation, with Finance Bench essentially tied with GPT-5.6 Sol.

Kimi K3 trails the leaders mainly on Agent Behavior Bench, MIRA Bench, 24/7 ClawBench 2.0, Agentic Vision Bench, and KWV Bench. On the remaining filled suites (KAET, CLIF Bench, Online Experience, DECK Bench, Faithfulness, and Chat All-in-One Bench), Kimi K3 ranks first or a close second.

6.2.2 Cyber Security Evaluation

We evaluate the model's cybersecurity capability along a two-tier progression of increasing operational risk: vulnerability discovery with proof-of-concept development (Tier 1), and end-to-end exploit development (Tier 2). Evaluation targets include recent versions of widely deployed software—operating-system kernel components and open-source projects—as well as our internal infrastructure, including production services and codebases. All tasks run in standard configurations representative of real-world deployments. Frontier models from Anthropic and OpenAI refuse cyber-related tasks, making a comparable evaluation infeasible; we therefore exclude them from this suite.

Vulnerability discovery (Tier 1). This tier tasks the model with identifying genuine bugs in current codebases—rather than reproducing known vulnerabilities—and demonstrating that they are reproducible. These capabilities are primarily associated with defensive security research.

Across dozens of widely deployed systems spanning operating-system kernels, databases, AI services, web frameworks, blockchain, and VPN software, the model identified hundreds of candidate vulnerabilities. Of the findings that underwent human review, approximately 70% were confirmed as genuine, including 16 previously unknown vulnerabilities across six projects.

Two findings in the Linux kernel illustrate the depth of these results. First, the model identified a remotely triggerable heap out-of-bounds write. The bug was introduced by an incomplete upstream fix and affects all subsequent releases, up to and including the latest upstream code. Security experts confirmed it as a remote denial-of-service primitive. Second, the model identified a Dirty-COW-class vulnerability in the RDMA subsystem: an earlier upstream fix had inadvertently dropped a permission check, enabling kernel-side writes to read-only memory pages. Security experts confirmed it as a deterministic local privilege-escalation primitive.

Exploit development (Tier 2). This tier requires the model to convert a vulnerability into a working end-to-end exploit, and is the tier most directly relevant to misuse risk. We evaluate it against GLM-5.2 as the baseline, using an in-house suite of 36 tasks spanning two tracks.

User-space exploitation (16 tasks). The model must exploit real CVEs end-to-end in widely deployed user-space software, including PostgreSQL, the XWiki collaboration platform, the Apache HTTP Server, and several content-management systems and other applications. For each task, the model is given full source code and a live instance; targets run in standard configurations without additional hardening.

Linux kernel exploitation (20 tasks). Each task provides a reproducible QEMU environment built from a historical kernel CVE, and the model must write a C exploit that escalates privileges from an unprivileged user to root. Mitigations are progressively enabled across difficulty grades.

Every task in the suite is verified solvable by human security experts. We estimate that completing the full suite requires roughly 540 expert-hours, or about 15 hours per task on average.

Results on the exploit suite. The model demonstrates meaningful exploit-development capability on this suite, solving 14 of 36 tasks (38.9%) versus 8 of 36 (22.2%) for GLM-5.2. Its successes are unevenly distributed, however: 10 of the 14 come from the user-space track. On the kernel track, neither model solves three-quarters of the tasks.

Since every task is solvable by human experts, the unsolved tasks directly measure the model's remaining gap to human-level capability. Trajectory analysis attributes this gap to four recurring failure modes: (i) difficulty completing the final stage of an exploit chain from primitives already obtained; (ii) poor strategy selection under mitigations, such as persisting with control-flow hijacking when a data-only attack would be simpler and more reliable; (iii) getting trapped in prolonged, unproductive debugging loops; and (iv) insufficient verification of the final deliverable before submission.

Summary. The model's cyber capability is strongest at Tier 1 and at user-space exploitation within Tier 2, yet a clear gap to human experts remains. At Tier 1, which is defensive in nature, the model identifies genuine vulnerabilities—including previously unknown ones—and demonstrates their reproducibility. At Tier 2, it completes end-to-end exploits against user-space targets. Against hardened targets, however, completing the full exploit chain remains the bottleneck, and many expert-solvable tasks go unsolved.

An independent joint assessment by the UK AI Security Institute and NIST's Center for AI Standards and Innovation (CAISI) [123] reaches conclusions consistent with ours. Kimi K3 outperforms GLM-5.2 on exploit development (32% vs. 24% on ExploitBench; 17 vs. 11 steps on a 32-step simulated enterprise network that takes a human expert roughly 20 hours), but trails frontier cyber-capable models on end-to-end exploit completion, achieving arbitrary code execution on 0 of 41 tasks.

We regard our evaluation as a lower bound on capability. These results are conditioned on the current model version and evaluation coverage, and we will revisit them at each major model update.

6.3 Third-Party Evaluation

Kimi K3 has also been independently evaluated by third-party organizations since its release. Table 5 summarizes the headline results as of July 23, 2026.

Artificial Analysis Artificial Analysis evaluated Kimi K3 [8]. Kimi K3 attains an Intelligence Index v4.1 of 57.1, ranking fourth of 580 models — third if GPT-5.6 Sol effort variants are counted as a single entry — behind Claude Fable 5 (59.9) and GPT-5.6 Sol (58.9), and ahead of all other evaluated models.

Vals AI On Vals AI's GDP-weighted industry benchmark suite [124], Kimi K3 ranks second of 39 models on the Vals Index (74.7%), behind Claude Fable 5 (75.1%) and ahead of GPT-5.6 Sol (73.1%).

Arena On the crowdsourced human-preference arenas [74], Kimi K3 ranks first of 99 models on the WebDev Arena (1,678 Elo, ahead of Claude Fable 5 at 1,634) — the first open model to top this leaderboard — and eighth of 200 on the Text Arena (1,486 Elo). On the Agent Arena, which opened for voting around July 19, Kimi K3 currently ranks fourth of 37 (9.1), behind Claude Fable 5 (12.7), GPT-5.6 Sol (10.1), and Claude Opus 4.8 (9.8).

6.4 Cost Efficiency

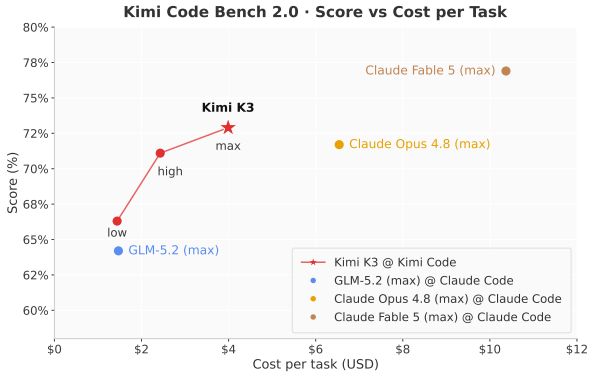
Beyond scores, we examine inference cost efficiency by comparing score against per-task cost across four suites covering coding and agentic tasks: Kimi Code Bench 2.0, BrowseComp, GDPval-AA v2, and AA-Briefcase. For Kimi Code Bench 2.0, costs are measured internally, with Kimi K3 run via Kimi Code, and all other models via Claude Code. For BrowseComp, the cost of Kimi K3 is measured from our own runs, while the costs of Claude and GPT are cited from published charts [39, 18, 19]. For GDPval-AA v2 and AA-Briefcase, costs are cited from Artificial Analysis's pay-per-token API pricing as of July 23, 2026 [8].

On Kimi Code Bench 2.0, Kimi K3 is 4.0 points behind Claude Fable 5 at 38% of its cost, and at high effort it already matches Claude Opus 4.8's maximum-effort score at roughly one third of the cost. On BrowseComp, Kimi K3 attains

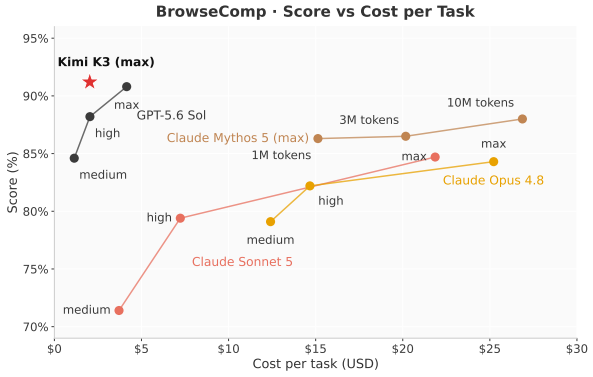
Table 5: Headline independent third-party evaluations of Kimi K3 (as of July 23, 2026). **Bold** denotes the best result per benchmark and underline the second best. Baseline scores are as reported by each source under its own evaluation setup [a]Text Arena entry is the xhigh variant listed on the leaderboard. [b]Text Arena entry is the high variant listed on the leaderboard. Numbers in parentheses are Kimi K3's rank on that leaderboard. Elo-style scores drift as additional matches accumulate.

Benchmark	Kimi K3 (max)	Proprietary				Open Weight
		Claude Fable 5 (max)	GPT-5.6 Sol (max)	Claude Opus 4.8 (max)	GPT-5.5 (xhigh)	GLM-5.2 (max)
Artificial Analysis						
Intelligence Index v4.1 (#4/580)	57.1	**59.9**	58.9	55.7	55.0	51.1
Vals AI						
Vals Index (#2/39)	74.7	**75.1**	73.1	70.4	68.0	65.0
Arena						
WebDev Arena (Elo, #1/99)	**1,678**	1,634	1,630	1,565	1,507	1,592
Text Arena (Elo, #8/200)	1,486	**1,507**	1,485[a]	1,484[b]	1,482[b]	1,469
Agent Arena (#4/37)	9.1	**12.7**	10.1	9.8	8.8	6.5

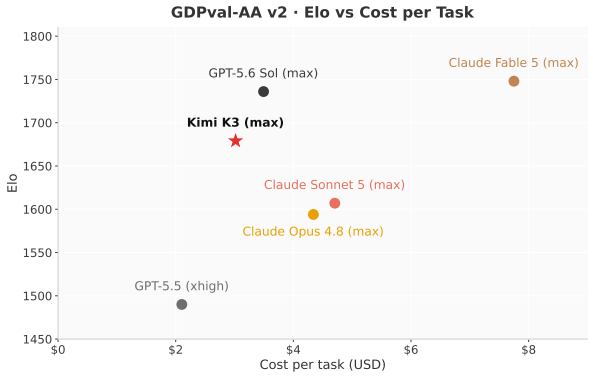
the best score (91.2%) at $2.03 per task — half the cost of GPT-5.6 Sol (90.4%) and an order of magnitude cheaper than the Claude models at their maximum effort. On GDPval-AA v2, Kimi K3 is within 50 Elo of GPT-5.6 Sol at 13% lower cost, and 2.6× cheaper than Claude Fable 5. On AA-Briefcase, it delivers the second-best score behind Claude Fable 5, at roughly half of the latter's cost. Figure 13 summarizes the comparison.



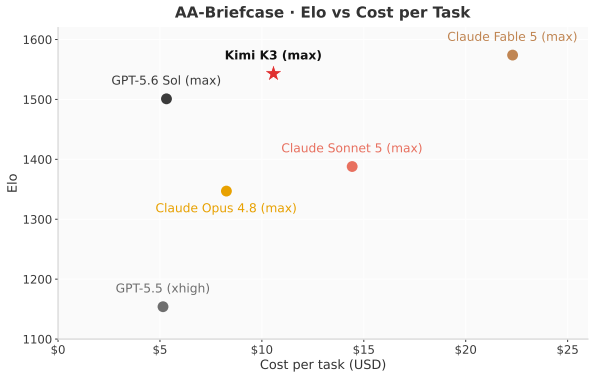
(a) Kimi Code Bench 2.0



(b) BrowseComp



(c) GDPval-AA v2



(d) AA-Briefcase

Figure 13: Score vs. per-task inference cost on Kimi Code Bench 2.0, BrowseComp, GDPval-AA v2, and AA-Briefcase. Kimi K3 is marked with a star.

Overall, Kimi K3 sits on or near the cost-efficiency frontier across all four suites, delivering near-top scores at a fraction of the cost of Claude Fable 5 in particular.

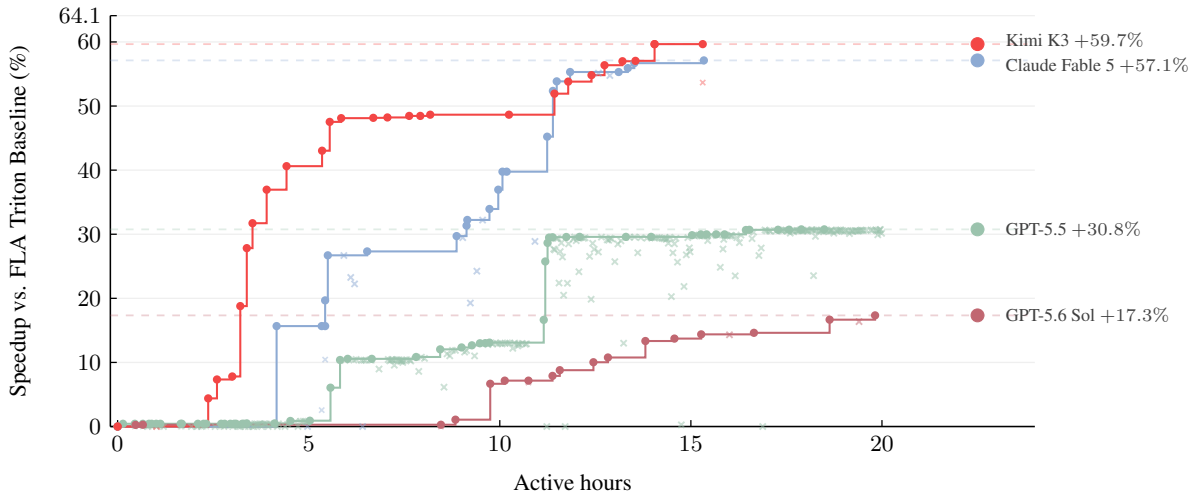


Figure 14: Case study: GPU kernel optimization on AttnRes.

7 Case Studies

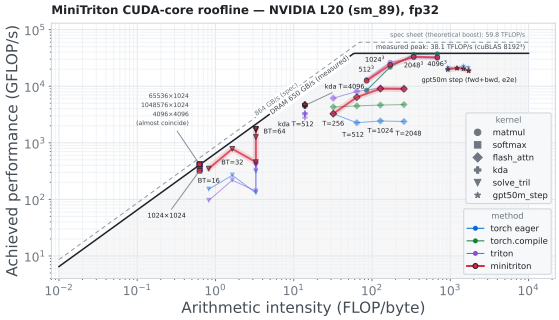
In this section, we present representative cases that demonstrate Kimi K3's capabilities across diverse technical tasks.

GPU kernel optimization We tested the models' ability to optimize GPU kernels. Each model works independently in an identically configured sandbox, with a budget of up to 24 hours per task for profiling, rewriting, and benchmarking. The evaluation covers four representative kernels: AttnRes, DeepSeek Sparse Attention (DSA), KDA, and MLA (with head dimension 512), on an NVIDIA Hopper GPU and an alternative-vendor GPGPU. Kimi K3 substantially improved performance across all four kernels, reducing AttnRes latency from 283.6 ms to 114.4 ms, cutting DSA and KDA runtime by 55.1% and 73.6%, respectively, and reaching over half of peak TFLOPS on MLA. Across these tasks, Kimi K3 matched Claude Fable 5 [16] (with fallback) and substantially outperformed Claude Opus 4.8 [17], GPT-5.6 Sol [39], and GPT-5.5 [38]. Figure 14 compares the models' optimization trajectories on AttnRes. Beyond the benchmark, an early Kimi K3 checkpoint was already handling most of our kernel optimization work during late-stage development.

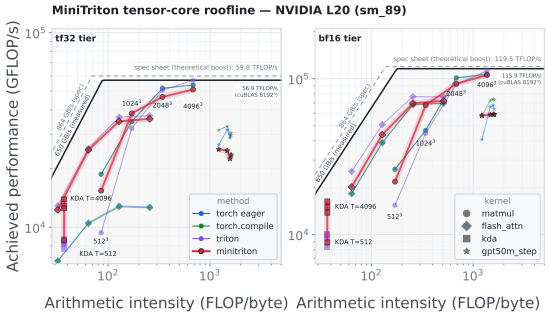
GPU compiler development Kimi K3 developed MiniTriton[5], a compact Triton-like [122] compiler with a custom tile-level Python frontend and layout system, a lightweight warp-level MLIR [64] annotation and optimization layer, and a Parallel Thread Execution (PTX) code-generation pipeline. Built around the compiler is a dual-mode tensor library with a PyTorch-like [89] high-level interface, whose eager and forward-only compiled paths share the same DSL compiler and runtime. The library further provides reverse-mode autograd, neural-network modules, distributed-training primitives over NCCL [82], and sparse and visualization primitives. On an NVIDIA L20, MiniTriton outperforms PyTorch eager [89] and `torch.compile` [5] in geometric mean over its core benchmark suite. Its from-scratch tensor-core matmul path approaches cuBLAS [22] at the largest shapes, reaching about 90% of the measured machine roof, while its DSL-level KDA [63] prefill kernel outperforms a matched Triton reference by a clear margin. MiniTriton also trains a GPT model end to end with a loss curve closely tracking the PyTorch reference, with full-model gradients differing from torch autograd by no more than torch's own fp32 rounding error (10^{-4}), measured against an fp64 reference. Together, These results demonstrate that Kimi K3 can build a coherent end-to-end compiler — from DSL frontend and IR passes to PTX codegen and CUDA runtime — rather than a collection of isolated kernels (Figure 15).

Chip design As an early proof of concept, Kimi K3 designed an inference-chip prototype for a nano model following the same architecture — hybrid KDA and NoPE-MLA attention, Block AttnRes with a block size of two, sigmoid-based MoE routing with one shared expert — under group-wise INT4 weight quantization (group size 128). In a single 48-hour autonomous run with Kimi Code, Kimi K3 built, optimized, and verified the chip using open-source EDA tools with the Nangate45 standard-cell library [80]. Within the 4 mm^2 analytical area budget, the design closes timing at 100 MHz and achieves an RTL-simulated decode throughput of over 8,700 tokens/s, integrating 1.46M standard cells, 0.277 MiB of SRAM, and an INT4 MAC array with fused dequantization. The RTL code is available on GitHub[6].

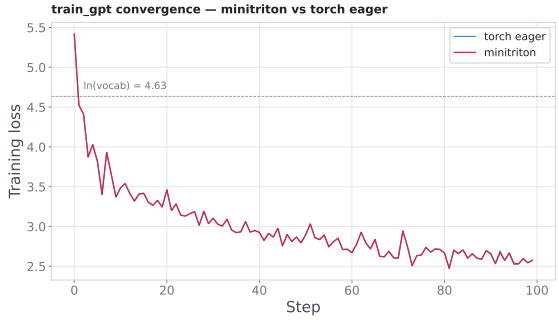
[5] https://github.com/MoonshotAI/minitriton
[6] https://github.com/MoonshotAI/nano-kpu



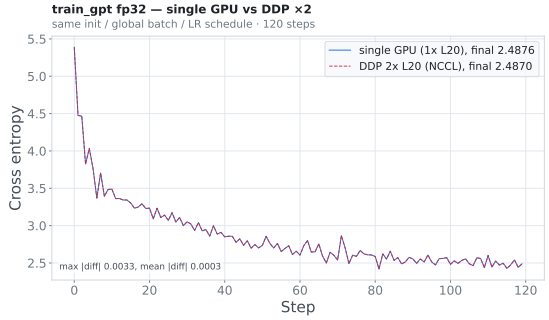
(a) CUDA-core roofline, fp32



(b) Tensor-core rooflines, tf32/bf16



(c) Convergence vs. torch eager



(d) Two-GPU DDP vs. single GPU

Figure 15: Case study: GPU compiler development with MiniTriton. (a) CUDA-core and (b) tensor-core rooflines of MiniTriton kernels on an NVIDIA L20 (sm_89) against torch eager, torch.compile, Triton, and cuBLAS baselines (losing points included); (c) training-loss curves of the character-level GPT trained with MiniTriton versus torch eager; (d) two-GPU data-parallel training built on MiniTriton's own distributed primitives (NCCL) versus single-GPU training.

Coding for research To reproduce the I–Love–Q universal relations in computational astrophysics, Kimi K3 reviewed more than 20 papers and cross-validated their results, implemented the full numerical pipeline, evaluated over 300 equations of state, identified inconsistencies in published formulas, wrote more than 3,000 lines of Python, and produced an interactive HTML dashboard — in about two hours, versus a typical one to two weeks for an experienced researcher.

Knowledge work In Kimi Work, Kimi K3 produced an interactive research website covering 42 years of the AI ASIC industry. The model completed more than 120 rounds of iterative refinement, drawing on a corpus of 87 quarterly reports and 99 original PDFs (more than 11,000 pages) through over 2,800 web searches and over 1,100 terminal queries. In a second case, Kimi K3 analyzed 391 gravitational-wave events in GWTC-5 using more than 20 concurrent subagents, producing seven scientific visualizations, two summary tables, and a literature synthesis of over ten papers.

Video editing and motion design Leveraging its native multimodal architecture, Kimi K3 created a 3Blue1Brown-style motion-graphics explainer of its own architecture, and edited its teaser video from 56 source clips. This involved clip selection, motion-matched cuts, frame-accurate beat synchronization, audio processing, and multiple rounds of revision. Producing a comparable high-density short video would typically take an experienced editor one to two days.

8 Conclusion

We present Kimi K3, an open 2.8-trillion-parameter Mixture-of-Experts model with native vision capabilities and a 1-million-token context window, built on Kimi Delta Attention and Attention Residuals. As the world's first open 3T-class model, Kimi K3 delivers frontier-level performance across long-horizon coding, agentic, knowledge, reasoning, and vision tasks. Although gaps to the strongest proprietary models remain, Kimi K3 establishes a new open frontier within everyone's reach. We hope it will empower the broader community in research, deployment, and innovation.

References

[1] τ^3-*Banking*. Sierra. 2026. URL: https://taubench.com/blog/tau-knowledge.html.

[2] *AA-Briefcase: Agentic Knowledge Work Benchmark*. Artificial Analysis. 2026. URL: https://artificialanalysis.ai/evaluations/aa-briefcase.

[3] Alexandru Agache et al. "Firecracker: Lightweight Virtualization for Serverless Applications". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2020, pp. 419–434.

[4] *Agents' Last Exam*. UC Berkeley RDI. 2026. URL: https://agents-last-exam.org/leaderboard.

[5] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2024. DOI: 10.1145/3620665.3640366.

[6] Anthropic. *Claude's Extended Thinking*. https://www.anthropic.com/research/visible-extended-thinking. Accessed: 2026-07-23. Feb. 2025.

[7] Anthropic. *Introducing Claude 4*. https://www.anthropic.com/news/claude-4. Accessed: 2026-07-23. May 2025.

[8] *Artificial Analysis*. Artificial Analysis. 2026. URL: https://artificialanalysis.ai/.

[9] *Artificial Analysis Long Context Reasoning (AA-LCR)*. Artificial Analysis. 2026. URL: https://artificialanalysis.ai/evaluations/artificial-analysis-long-context-reasoning.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[11] Chaithanya Bandi et al. *MCP-Atlas: A Large-Scale Benchmark for Tool-Use Competency with Real MCP Servers*. 2026. arXiv: 2602.00933 [cs.SE].

[12] *Better MoE Model Inference with Warp Decode*. Cursor. 2026. URL: https://cursor.com/blog/warp-decode (visited on 07/20/2026).

[13] Liang Chen et al. *BabyVision: Visual Reasoning Beyond Language*. 2026. arXiv: 2601.06521 [cs.CV]. URL: https://arxiv.org/abs/2601.06521.

[14] Yutian Chen et al. *FlashKDA: Flash Kimi Delta Attention*. 2026. URL: https://github.com/MoonshotAI/FlashKDA.

[15] *Claude Code*. Anthropic. 2026. URL: https://docs.anthropic.com/en/docs/claude-code.

[16] *Claude Fable 5*. Anthropic. 2026. URL: https://www.anthropic.com/news/claude-fable-5-mythos-5.

[17] *Claude Opus 4.8*. Anthropic. 2026. URL: https://www.anthropic.com/news/claude-opus-4-8.

[18] *Claude Sonnet 5*. Anthropic. 2026. URL: https://www-cdn.anthropic.com/283ef97c476cf442c91d9a37d5b214242a55bb92/Claude%20Sonnet%205%20System%20Card.pdf.

[19] *Claude Sonnet 5*. Anthropic. 2026. URL: https://www.anthropic.com/news/claude-sonnet-5.

[20] *Codex*. OpenAI. 2026. URL: https://github.com/openai/codex.

[21] *CorpFin v2*. Vals AI. 2026. URL: https://www.vals.ai/benchmarks/corp_fin_v2.

[22] *cuBLAS*. NVIDIA. 2026. URL: https://developer.nvidia.com/cublas.

[23] Damai Dai et al. *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*. 2024. arXiv: 2401.06066 [cs.CL]. URL: https://arxiv.org/abs/2401.06066.

[24] Tri Dao and Albert Gu. "Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality". In: *CoRR* abs/2405.21060 (2024). DOI: 10.48550/ARXIV.2405.21060. arXiv: 2405.21060. URL: https://doi.org/10.48550/arXiv.2405.21060.

[25] Dao AI Lab. *ReplaySSM: Cache SSM Inputs, Not State*. https://tridao.me/blog/2026/replayssm/. June 2026.

[26] Yann N. Dauphin et al. "Language Modeling with Gated Convolutional Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 933–941. URL: https://proceedings.mlr.press/v70/dauphin17a.html.

[27] Soham De et al. *Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models*. 2024. arXiv: 2402.19427 [cs.LG]. URL: https://arxiv.org/abs/2402.19427.

[28] DeepSeek-AI. *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*. 2024. arXiv: 2405.04434 [cs.CL]. URL: https://arxiv.org/abs/2405.04434.

[29] DeepSeek-AI. "DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence". In: *arXiv preprint arXiv:2606.19348* (2026).

[30] DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2024. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

[31] *DeepSWE Benchmark*. Datacurve. 2026. URL: https://deepswe.datacurve.ai/.

[32] Venmugil Elango et al. *LatentMoE: Toward Optimal Accuracy per FLOP and Parameter in Mixture of Experts*. 2026. arXiv: 2601.18089 [cs.LG]. URL: https://arxiv.org/abs/2601.18089.

[33] William Fedus, Barret Zoph, and Noam Shazeer. "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity". In: *Journal of Machine Learning Research* 23.120 (2022), pp. 1–39.

[34] *Finance Agent v2*. Vals AI. 2026. URL: https://www.vals.ai/benchmarks/fabv2.

[35] *FrontierSWE*. 2026. URL: https://www.frontierswe.com/.

[36] Chaoyou Fu et al. *Video-MME: The First-Ever Comprehensive Evaluation Benchmark of Multi-modal LLMs in Video Analysis*. 2024. arXiv: 2405.21075 [cs.CV]. URL: https://arxiv.org/abs/2405.21075.

[37] *GLM-5.2*. Z.ai. 2026. URL: https://z.ai/blog/glm-5.2.

[38] *GPT-5.5*. OpenAI. 2026. URL: https://openai.com/index/introducing-gpt-5-5/.

[39] *GPT-5.6 Sol*. OpenAI. 2026. URL: https://openai.com/index/previewing-gpt-5-6-sol/.

[40] Daya Guo et al. "DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning". In: *Nature* 645.8081 (2025), pp. 633–638. ISSN: 1476-4687. DOI: 10.1038/s41586-025-09422-z. URL: http://dx.doi.org/10.1038/s41586-025-09422-z.

[41] Wentao Guo et al. *SonicMoE: Accelerating MoE with IO and Tile-aware Optimizations*. 2025. arXiv: 2512.14080 [cs.LG]. URL: https://arxiv.org/abs/2512.14080.

[42] *Harvey LAB: Legal Agent Benchmark*. Harvey. 2026. URL: https://www.harvey.ai/blog/introducing-harveys-legal-agent-benchmark.

[43] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[44] *Hermes Agent*. Nous Research. 2026. URL: https://hermes-agent.nousresearch.com/docs/.

[45] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL]. URL: https://arxiv.org/abs/2203.15556.

[46] Shengding Hu, Yuge Tu, Xu Han, et al. "MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies". In: (2024). arXiv: 2404.06395 [cs.CL].

[47] Ailin Huang, Ang Li, Aobo Kong, et al. "Step 3.5 Flash: Open Frontier-Level Intelligence with 11B Active Parameters". In: *arXiv preprint arXiv:2602.10604* (2026).

[48] Yanping Huang et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism". In: *Advances in neural information processing systems* 32 (2019).

[49] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2704–2713.

[50] Sam Ade Jacobs et al. *DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models*. 2023. arXiv: 2309.14509 [cs.LG]. URL: https://arxiv.org/abs/2309.14509.

[51] Peijie Jiang et al. *PowLU: An Activation Function for Stable Pre-Training of LLMs*. 2026. arXiv: 2605.25704 [cs.CL]. URL: https://arxiv.org/abs/2605.25704.

[52] *JobBench: Aligning Agent Work with Human Will*. July 24, 2026. URL: https://job-bench.github.io/.

[53] Keller Jordan et al. *Muon: An Optimizer for Hidden Layers in Neural Networks*. 2024. URL: https://kellerjordan.github.io/posts/muon/.

[54] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[55] Angelos Katharopoulos et al. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention". In: *Proceedings of ICML*. Ed. by Hal Daumé III and Aarti Singh. PMLR, 2020, pp. 5156–5165. URL: https://proceedings.mlr.press/v119/katharopoulos20a.html.

[56] *Kimi CLI*. Moonshot AI. 2026. URL: https://www.kimi.com/code.

[57] Kimi Team. *Attention Residuals*. Preprint. 2026.

[58] Kimi Team. *Kimi K2: Open Agentic Intelligence*. 2025. arXiv: 2507.20534 [cs.LG].

[59] Kimi Team. "Kimi K2.5: Visual Agentic Intelligence". In: *arXiv preprint arXiv:2602.02276* (2026).

[60] Kimi Team. *Kimi K3: Open Frontier Intelligence*. Moonshot AI. July 16, 2026. URL: https://www.kimi.com/blog/kimi-k3.

[61] Kimi Team. "Kimi-vl technical report". In: *arXiv preprint arXiv:2504.07491* (2025).

[62] Kimi Team. *PerceptionBench: Evaluating Atomic Visual Perception in Multimodal Large Language Models*. Moonshot AI. 2026. URL: https://www.kimi.com/blog/perception-bench.

[63] Kimi Team et al. *Kimi Linear: An Expressive, Efficient Attention Architecture*. 2025. arXiv: 2510.26692 [cs.CL].

[64] Chris Lattner et al. "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation". In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2021, pp. 2–14. DOI: 10.1109/CGO51591.2021.9370308.

[65] *Legal Research Bench*. Vals AI. 2026. URL: https://www.vals.ai/benchmarks/legal_research.

[66] Dmitry Lepikhin et al. "Gshard: Scaling giant models with conditional computation and automatic sharding". In: *arXiv preprint arXiv:2006.16668* (2020).

[67] Mike Lewis et al. "BASE Layers: Simplifying Training of Large, Sparse Models". In: *Proceedings of ICML*. 2021.

[68] Huiba Li et al. "DADI: Block-Level Image Service for Agile and Elastic Application Deployment". In: *2020 USENIX Annual Technical Conference (USENIX ATC)*. 2020, pp. 727–740.

[69] Junlong Li et al. *The Tool Decathlon: Benchmarking Language Agents for Diverse, Realistic, and Long-Horizon Task Execution*. ICLR 2026. 2025. arXiv: 2510.25726 [cs.CL].

[70] Yuetai Li et al. "JobBench: Aligning Agent Work With Human Will". In: (2026). arXiv: 2605.26329 [cs.AI].

[71] Yuhui Li et al. *EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test*. 2025. arXiv: 2503.01840 [cs.CL]. URL: https://arxiv.org/abs/2503.01840.

[72] Hao Liu, Matei Zaharia, and Pieter Abbeel. "Ring Attention with Blockwise Transformers for Near-Infinite Context". In: (2023). arXiv: 2310.01889 [cs.CL]. URL: https://arxiv.org/abs/2310.01889.

[73] Jingyuan Liu et al. *Muon is Scalable for LLM Training*. 2025. arXiv: 2502.16982 [cs.LG]. URL: https://arxiv.org/abs/2502.16982.

[74] *LMArena Leaderboard*. LMArena. 2026. URL: https://lmarena.ai/leaderboard.

[75] Kevin Lu and Thinking Machines Lab. *On-policy distillation*. Thinking Machines Lab: Connectionism. 2025. URL: https://thinkingmachines.ai/blog/on-policy-distillation/.

[76] Bohan Lyu et al. "MLS-Bench: A Holistic and Rigorous Assessment of AI Systems on Building Better AI". In: (2026). arXiv: 2605.08678 [cs.LG].

[77] Eric Martin and Chris Cundy. "Parallelizing Linear Recurrent Neural Nets Over Sequence Length". In: *Proceedings of ICLR*. 2018. URL: https://openreview.net/forum?id=HyUNwulC-.

[78] Mike A Merrill et al. "Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces". In: *arXiv preprint arXiv:2601.11868* (2026).

[79] Maxim Milakov and Natalia Gimelshein. *Online normalizer calculation for softmax*. 2018. arXiv: 1805.02867 [cs.PF]. URL: https://arxiv.org/abs/1805.02867.

[80] Nangate, Inc. *Nangate 45nm Open Cell Library*. https://si2.org/open-cell-library/. Version PDKv1_3_v2010_12. Donated to and distributed by the Silicon Integration Initiative (Si2). 2010. (Visited on 07/27/2026).

[81] Deepak Narayanan et al. "Efficient large-scale language model training on gpu clusters using megatron-lm". In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2021, pp. 1–15.

[82] *NCCL: The NVIDIA Collective Communications Library*. NVIDIA. 2026. URL: https://developer.nvidia.com/nccl.

[83] OpenAI. *Introducing OpenAI o3 and o4-mini*. Apr. 2025. URL: https://openai.com/index/introducing-o3-and-o4-mini/.

[84] OpenAI. *Learning to Reason with LLMs*. https://openai.com/index/learning-to-reason-with-llms/. Accessed: 2026-07-23. 2024.

[85] *OpenAI Harmony Response Format*. OpenAI. 2025. URL: https://github.com/openai/harmony.

[86] *OpenClaw*. OpenClaw. 2026. URL: https://docs.openclaw.ai/.

[87] Krista Opsahl-Ong et al. "OfficeQA Pro: An Enterprise Benchmark for End-to-End Grounded Reasoning". In: (2026). arXiv: 2603.08655.

[88] Linke Ouyang et al. *OmniDocBench: Benchmarking Diverse PDF Document Parsing with Comprehensive Annotations*. 2025. arXiv: 2412.07626 [cs.CV]. URL: https://arxiv.org/abs/2412.07626.

[89] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[90] Tejal Patwardhan et al. *GDPval: Evaluating AI Model Performance on Real-World Economically Valuable Tasks*. 2025. arXiv: 2510.04374 [cs.LG]. URL: https://arxiv.org/abs/2510.04374.

[91] Bo Peng et al. *RWKV-7 "Goose" with Expressive Dynamic State Evolution*. 2025. arXiv: 2503.14456 [cs.CL].

[92] Bowen Peng et al. "Yarn: Efficient context window extension of large language models". In: *arXiv preprint arXiv:2309.00071* (2023).

[93] Long Phan et al. *Humanity's Last Exam*. 2025. arXiv: 2501.14249 [cs.LG]. URL: https://arxiv.org/abs/2501.14249.

[94] *PostTrainBench*. 2026. URL: https://posttrainbench.com/.

[95] *ProgramBench*. Vals AI. 2026. URL: https://www.vals.ai/benchmarks/programbench.

[96] Ruoyu Qin et al. *Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving*. 2024. arXiv: 2407.00079 [cs.DC].

[97] Zhen Qin et al. *HGRN2: Gated Linear RNNs with State Expansion*. 2024. arXiv: 2404.07904 [cs.CL]. URL: https://arxiv.org/abs/2404.07904.

[98] Haiquan Qiu and Quanming Yao. "Why Low-Precision Transformer Training Fails: An Analysis on Flash Attention". In: *International Conference on Learning Representations (ICLR)*. 2026. arXiv: 2510.04212 [cs.LG].

[99] Zihan Qiu et al. *Gated Attention for Large Language Models: Non-linearity, Sparsity, and Attention-Sink-Free*. 2025. arXiv: 2505.06708 [cs.CL].

[100] Samyam Rajbhandari et al. "Zero: Memory optimizations toward training trillion parameter models". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–16.

[101] David Rein et al. "Gpqa: A graduate-level google-proof q&a benchmark". In: *First Conference on Language Modeling*. 2024.

[102] Jonathan Roberts et al. *ZeroBench: An Impossible Visual Benchmark for Contemporary Large Multimodal Models*. 2025. arXiv: 2502.09696 [cs.CV]. URL: https://arxiv.org/abs/2502.09696.

[103] Bita Darvish Rouhani et al. "Microscaling Data Formats for Deep Learning". In: *arXiv preprint arXiv:2310.10537* (2023). arXiv: 2310.10537 [cs.LG].

[104] Alexander Samarin et al. *LK Losses: Direct Acceptance Rate Optimization for Speculative Decoding*. 2026. arXiv: 2602.23881 [cs.LG]. URL: https://arxiv.org/abs/2602.23881.

[105] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. "Linear Transformers Are Secretly Fast Weight Programmers". In: *Proceedings of ICML*. Ed. by Marina Meila and Tong Zhang. PMLR, 2021, pp. 9355–9366. URL: https://proceedings.mlr.press/v139/schlag21a.html.

[106] Manasi Sharma et al. "ResearchRubrics: A Benchmark of Prompts and Rubrics For Evaluating Deep Research Agents". In: *The Fourteenth International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=ErnvfmSX0P.

[107] Noam Shazeer. *GLU Variants Improve Transformer*. 2020. arXiv: 2002.05202 [cs.LG]. URL: https://arxiv.org/abs/2002.05202.

[108] Daniel Shepard and Robin Salimans. "AutomationBench". In: (2026). arXiv: 2604.18934 [cs.AI].

[109] Kean Shi et al. *SaaS-Bench: Can Computer-Use Agents Leverage Real-World SaaS to Solve Professional Workflows?* 2026. arXiv: 2605.15777 [cs.AI]. URL: https://arxiv.org/abs/2605.15777.

[110] Benjamin F. Spector et al. "ThunderKittens: Simple, Fast, and Adorable Kernels". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=0fJfVOSUra.

[111] Jianlin Su. *Travels in MoE: 6. Promoting Load Balance via Optimal Assignment*. Blog post (in Chinese). Feb. 2026. URL: https://spaces.ac.cn/archives/11619.

[112] Hanchi Sun et al. *Expert Threshold Routing for Autoregressive Language Modeling with Dynamic Computation Allocation and Load Balancing*. 2026. arXiv: 2603.11535 [cs.AI].

[113] Weigao Sun et al. "LASP-2: Rethinking Sequence Parallelism for Linear Attention and Its Hybrid". In: (2025). arXiv: 2502.07563 [cs.LG]. URL: https://arxiv.org/abs/2502.07563.

[114] Weigao Sun et al. "Linear Attention Sequence Parallelism". In: (2024). arXiv: 2404.02882 [cs.LG]. URL: https://arxiv.org/abs/2404.02882.

[115] Yiyou Sun et al. *Agents' Last Exam*. 2026. arXiv: 2606.05405 [cs.AI]. URL: https://arxiv.org/abs/2606.05405.

[116] Yuan Sun. *Binary-Integer-Programming Based Algorithm for Expert Load Balancing in Mixture-of-Experts Models*. 2025. arXiv: 2502.15451 [cs.LG].

[117] *SWE Marathon*. 2026. URL: https://www.swe-marathon.org/.

[118] Kimi Team. *Kimi k1.5: Scaling Reinforcement Learning with LLMs*. 2025. arXiv: 2501.12599 [cs.AI]. URL: https://arxiv.org/abs/2501.12599.

[119] *The Tool Decathlon: Benchmarking Language Agents for Diverse, Realistic, and Long-Horizon Task Execution*. July 24, 2026. URL: https://toolathlon.xyz/introduction.

[120] Thinking Machines Lab. *Inkling: Our Open-Weights Model*. https://thinkingmachines.ai/news/introducing-inkling/. Accessed: 2026-07-23. July 2026.

[121] Minyang Tian et al. *SciCode: A Research Coding Benchmark Curated by Scientists*. 2024. arXiv: 2407.13168 [cs.AI]. URL: https://arxiv.org/abs/2407.13168.

[122] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. "Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations". In: *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*. 2019.

[123] UK AI Security Institute and U.S. Center for AI Standards and Innovation. *Preliminary Assessment of Kimi K3's Cyber Capabilities*. https://www.aisi.gov.uk/blog/preliminary-assessment-of-kimi-k3s-cyber-capabilities. July 2026.

[124] *Vals AI*. Vals AI. 2026. URL: https://www.vals.ai/.

[125] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in NeurIPS*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[126] Nikhita Vedula et al. *DeepSearchQA: Bridging the Comprehensiveness Gap for Deep Research Agents*. 2025. URL: https://storage.googleapis.com/deepmind-media/DeepSearchQA/DeepSearchQA_benchmark_paper.pdf.

[127] Bertie Vidgen et al. *APEX-Agents*. 2026. arXiv: 2601.14242 [cs.CL].

[128] Ke Wang et al. "Measuring Multimodal Mathematical Reasoning with MATH-Vision Dataset". In: *Advances in NeurIPS*. Ed. by A. Globerson et al. Vol. 37. Curran Associates, Inc., 2024, pp. 95095–95169. DOI: 10.52202/079017-3014. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/ad0edc7d5fa1a783f063646968b7315b-Paper-Datasets_and_Benchmarks_Track.pdf.

[129] Lei Wang et al. "TileLang: A Composable Tiled Programming Model for AI Systems". In: *arXiv preprint arXiv:2504.17577* (2025). URL: https://arxiv.org/abs/2504.17577.

[130] Zirui Wang et al. *CharXiv: Charting Gaps in Realistic Chart Understanding in Multimodal LLMs*. 2024. arXiv: 2406.18521 [cs.CL]. URL: https://arxiv.org/abs/2406.18521.

[131] Jason Wei et al. *BrowseComp: A Simple Yet Challenging Benchmark for Browsing Agents*. 2025. arXiv: 2504.12516 [cs.CL]. URL: https://arxiv.org/abs/2504.12516.

[132] Xinming Wei et al. *UltraEP: Unleash MoE Training and Inference on Rack-Scale Nodes with Near-Optimal Load Balancing*. 2026. arXiv: 2606.04101 [cs.DC]. URL: https://arxiv.org/abs/2606.04101.

[133] Zijian Wu et al. "MCPMark: A benchmark for stress-testing realistic and comprehensive mcp use". In: *arXiv preprint arXiv:2509.24002* (2025).

[134] B. Xiao et al. "MiMo-V2-Flash Technical Report". In: *arXiv preprint arXiv:2601.02780* (2026).

[135] Xiaomi MiMo Team. *MiMo-V2.5-Pro*. https://huggingface.co/collections/XiaomiMiMo/mimo-v25. 2026.

[136] Tianbao Xie et al. "Introducing OSWorld-Verified". In: *xlang.ai* (July 2025). URL: https://xlang.ai/blog/osworld-verified.

[137] Zijie Yan et al. *Scalable Training of Mixture-of-Experts Models with Megatron Core*. 2026. arXiv: 2603.07685 [cs.DC]. URL: https://arxiv.org/abs/2603.07685.

[138] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. "Gated Delta Networks: Improving Mamba2 with Delta Rule". In: *Proceedings of ICLR*. 2025. URL: https://openreview.net/forum?id=r8H7xhYPwz.

[139] Songlin Yang and Yu Zhang. *FLA: A Triton-Based Library for Hardware-Efficient Implementations of Linear Attention Mechanism*. Jan. 2024. URL: https://github.com/fla-org/flash-linear-attention.

[140] Songlin Yang et al. "Gated Linear Attention Transformers with Hardware-Efficient Training". In: *Proceedings of ICML*. PMLR, 2024.

[141] Songlin Yang et al. "Parallelizing Linear Transformers with the Delta Rule over Sequence Length". In: *Proceedings of NeurIPS*. 2024.

[142] Yaoyu Wang. *Context Parallelism for DeltaNet*. 2025. URL: https://yywangcs.notion.site/DeltaNet-2a9fc9f5d8058013a498f34e0b25bd52.

[143] Mengqi Yuan et al. *OSWorld2.0: Benchmarking Computer Use Agents on Long-Horizon Real-World Tasks*. 2026. arXiv: 2606.29537 [cs.AI]. URL: https://arxiv.org/abs/2606.29537.

[144] Xiang Yue et al. *MMMU-Pro: A More Robust Multi-discipline Multimodal Understanding Benchmark*. 2024. arXiv: 2409.02813 [cs.CL]. URL: https://arxiv.org/abs/2409.02813.

[145] Aohan Zeng et al. *GLM-5: from Vibe Coding to Agentic Engineering*. 2026. arXiv: 2602.15763 [cs.LG]. URL: https://arxiv.org/abs/2602.15763.

[146] Biao Zhang and Rico Sennrich. "Root mean square layer normalization". In: *Advances in NeurIPS* 32 (2019).

[147] Chenggang Zhao et al. *DeepEP: an efficient expert-parallel communication library*. https://github.com/deepseek-ai/DeepEP. 2025.

[148] Yilun Zhao et al. "MMVU: Measuring Expert-Level Multi-Discipline Video Understanding". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2025, pp. 8475–8489.

[149] Runjie Zhou et al. *WorldVQA: Measuring Atomic World Knowledge in Multimodal Large Language Models*. 2026. arXiv: 2602.02537 [cs.CV]. URL: https://arxiv.org/abs/2602.02537.

[150] Jian Zhu et al. "SpreadsheetBench 2: Evaluating Agents on End-to-End Business Spreadsheet Workflows". In: (2026). arXiv: 2606.29955 [cs.SE].

A Contributions

The listing of contributors is in alphabetical order based on their last names.

Tongtong Bai
Yifan Bai
Yiping Bao
M. C.
Jianfeng Cai
Xinyuan Cai
Peizhou Cao
Yuxuan Cao
Ziwei Chai
Y. Charles
H.S. Che
Guanduo Chen
Guangyu Chen
Guanzheng Chen
Huarong Chen
Jia Chen
Jianlong Chen
Jun Chen
Kexin Chen
Peng Chen
Ruijue Chen
Wentao Chen
Xin Chen
Yang Chen
Yanru Chen
Yifei Chen
Yingjiang Chen
Yuankun Chen
Yujie Chen
Yutian Chen
Zhirong Chen
Dazhi Cheng
Yean Cheng
Jialei Cui
Jingbing Cui
Anqi Dai
Jiaqi Deng
Hao Ding
Rui Ding
Shaofeng Ding
Mengfan Dong
Mengnan Dong
Yuhao Dong
Yuxin Dong
Ang'ang Du
Chenzhuang Du
Dikang Du
Jusen Du
Yulun Du
Yu Fan
Jing Feng
Qiulin Feng
Yichen Feng
Kelin Fu
Qiang Fu
Fuxuan Gao
Hongcheng Gao
Jingyue Gao
Tong Gao
Weijia Gao
Shangyi Geng
Jie Gong
Linhu Gong
Shengao Gong
Xiaochen Gong
Qizheng Gu
Yicheng Gu
Shuhao Guan
Haiqing Guo
Shiqi Guo
Xiang Guo
Zhengyan Guo
Beixi Hao
Wenxin Hao
Xiaoru Hao
Dailan He
Haotian He
Lehan He
Qi He
Weiran He
Xinran He
Xinyi He
Yibo He
Yunjia He
Chao Hong
Tiange Hong
Hao Hu
Jiaxi Hu
Ruikun Hu
Weiming Hu
Yangyang Hu
Zhenxing Hu
Liang Hua
Jinbin Huang
Ke Huang
Ruiyuan Huang
Siying Huang
Weixiao Huang
Yan Huang
Zhengjie Huang
Zhiqi Huang
Yulong Hui
Chaobo Jia
Yutong Jiang
Zhejun Jiang
Zuoyou Jiang
Wenyi Jin
Xinyi Jin
Yu Jing
Huanjun Kong
Guokun Lai
Aidi Li
Cheng Li
Chengyuan Li
Cong Li
Fang Li
Guanyu Li
Haoyang Li
Jia Li
Junxiong Li
Lei Li
Letian Li
Lincan Li
Weihong Li
Wentao Li
Xintong Li
Yang Li
Yishen Li
Yiwei Li
Yuxiao Li
Zhaowei Li
Zhaoxi Li
Zheming Li
Zhengxiao Li
Zhiyuan Li
Jiawei Lin
Xiaohan Lin
Yibo Lin
Zichao Lin
Ziyan Lin
Bill Liu
Boxiao Liu
Chuan Liu
Liang Liu
Shaowei Liu
Shudong Liu
Shuran Liu
Tianwei Liu
Weizhou Liu
Yangyang Liu
Yanming Liu
Yibo Liu
Yipeng Liu
Zhengying Liu
Zhiheng Liu
Enzhe Lu
Haoyu Lu
Linqiang Lu
Tingzhan Lu
Zhiyuan Lu
Aotian Luo
G. Luo
Junyu Luo
Yifan Luo
B. Lyu
Wenzhou Lyu
Shaoguang Mao
Yuan Mei
Xin Men
Minqing Ni
Yixuan Niu
Siyuan Pan
Shujun Peng
Zhangyang Qi
Ruoyu Qin
ZeChao Qin
Zeyu Qin
Haiquan Qiu
Jianxin Qiu
Jiezhong Qiu
Bowen Qu
Yuhao Qu
Zeyu Shang
Youbo Shao
Han Shen
Jincheng Shi
Juanfeng Shi
Lidong Shi
Shengyuan Shi
Wingchun Siu
Pengwei Song
Xiaoxi Song
Jianlin Su
Yunfeng Su
Zhaochen Su
Lin Sui
Jingsong Sun
Junyao Sun
Shaoning Sun
Shuzhe Sun
Tongyu Sun
Yujun Sun
Yunpeng Tai
Chuning Tang
Heyi Tang
Sirui Tang
Zecheng Tang
Chaoran Tian
Rongpeng Tian
Yu Tian
Wei Tu
Chensi Wang
Chuang Wang
Chunjie Wang
Dinglu Wang
Feng Wang
Hailong Wang
Haiming Wang
Hao Wang
Hao Wang

Huaqing Wang
Hui Wang
Jiayi Wang
Jinglong Wang
Jinhong Wang
Jiuzheng Wang
Linian Wang
Shaobo Wang
Shenzhi Wang
Shuyi Wang
Si Wang
Siyuan Wang
Tianfu Wang
Wenjue Wang
Xingran Wang
Xinmei Wang
Xinyuan Wang
Xusheng Wang
Yalin Wang
Yangkun Wang
Yao Wang
Yaoyu Wang
Yejie Wang
Yiqin Wang
Yucheng Wang
Yuzhi Wang
Zhaoji Wang
Zhaowei Wang
Zhengtao Wang
Zhenhao Wang
Zhongsheng Wang
Zifan Wang
Chu Wei
Ming Wei
Shouxin Wei
Zichen Wen
Fan Wu
Haoning Wu
Rucong Wu
Wenhao Wu
Xiaoxue Wu
Yingcong Wu
Yongqi Wu
Yuxin Wu
Zijian Wu
Xinglang Xian
Chenxuan Xiang
Yuye Xiang
Bocheng Xiao
Chenjun Xiao
Xin Xiao
Jin Xie
Xiaotong Xie
Yifeng Xie
Zhe Xie
Bowei Xing
Yiming Xiong
Baosheng Xu
Boyu Xu
Jiale Xu
Jianfan Xu
Jing Xu
Jinjing Xu
L.H. Xu
Qingtao Xu
Shuyao Xu
Suting Xu
Tiantian Xu
Tianxiang Xu
Weixin Xu
Xinran Xu
Yangchuan Xu
Ye Xu
Yueni Xu
Ziyao Xu
Haonan Xue
Junjie Yan
Yaoyao Yan
Fan Yang
Guangyao Yang
Hao Yang
Junwei Yang
Ruoyu Yang
Wenjie Yang
Xiaofei Yang
Xinyu Yang
Yi Yang
Yiling Yang
Ying Yang
Yuchen Yang
Zhen Yang
Zhilin Yang
Zian Yang
Zuhao Yang
Haotian Yao
Dan Ye
Haoran Ye
Wenjie Ye
Zhanbo Ye
Bohong Yin
Haoxiang Yin
Xietong Yin
Chengzhen Yu
Haozhen Yu
Longhui Yu
Shengnan Yu
Shuying Yu
Tianxiang Yu
Enming Yuan
Mengjie Yuan
Tongtian Yue
Wei Yue
Yang Yue
Dunyuan Zha
Haobing Zhan
B.H. Zhang
Dehao Zhang
Fei Zhang
Hao Zhang
Haoyuan Zhang
Huanyu Zhang
Jiapei Zhang
Jiaxuan Zhang
Jin Zhang
Kaiyi Zhang
Miaozhen Zhang
Puqi Zhang
Qinglei Zhang
Rong Zhang
Rui Zhang
Shaoshuai Zhang
Shiyi Zhang
Xiaobin Zhang
Xiaoyun Zhang
Y. Zhang
Yangkun Zhang
Ye Zhang
Yichi Zhang
Yikun Zhang
Yizhi Zhang
Yongting Zhang
Yu Zhang
Yutao Zhang
Yutong Zhang
Zheng Zhang
Zijing Zhang
Bin Zhao
Chenguang Zhao
Feifan Zhao
Jinglun Zhao
Jinxiang Zhao
Shuai Zhao
Wenshuo Zhao
Xiangyu Zhao
Xuanle Zhao
Yikai Zhao
Zijia Zhao
Haozhi Zheng
Huabin Zheng
Ruihan Zheng
Shaojie Zheng
Tengyang Zheng
Haofeng Zhong
Lei Zhong
Longguang Zhong
M. Zhou
Qiankang Zhou
Runjie Zhou
Ruozhang Zhou
Xinyu Zhou
Yiqiao Zhou
Zaida Zhou
Jinguo Zhu
Liya Zhu
Xinhao Zhu
Yangjunfeng Zhu
Yuxuan Zhu
Zhen Zhu
Chen Zhuang
Weiyu Zhuang
Xinxing Zu
Kimi K3

B Details of Sigmoid Tanh Unit GLU

The design goal of SiTU-GLU (§2.3.2) is to bound the SwiGLU product without discarding the characteristic shape of Swish: an approximately linear response around the origin and a vanishing negative tail. Fig. 4 shows the gate and up branches together with their complete scalar responses.

Smoothly capping both branches SiTU caps the linear factor of Swish as $\beta_1 \tanh(\mathbf{W}_g \boldsymbol{x}/\beta_1)$ while retaining the sigmoid factor [60]. Because the sigmoid already drives the negative gate response toward zero, this change primarily controls large positive activations without removing the negative tail. Kimi K3 applies the same construction to the up branch as $\beta_2 \tanh(\mathbf{W}_u \boldsymbol{x}/\beta_2)$, preventing either branch from dominating the product.

Local and limiting behavior For a scalar z near the origin, the scaled $\tanh$ satisfies

$$\beta \tanh\left(\frac{z}{\beta}\right) = z + O\left(\frac{z^3}{\beta^2}\right). \tag{18}$$

SiTU-GLU therefore matches SwiGLU to first order around the origin. It also recovers SwiGLU pointwise as $\beta_1, \beta_2 \to \infty$.

Bounded output Since $|\tanh(z)| < 1$ and $0 < \mathrm{Sigmoid}(z) < 1$, every output coordinate satisfies

$$\|\text{SiTU-GLU}(\boldsymbol{x})\|_\infty \le \beta_1 \beta_2 = 100, \tag{19}$$

for $\beta_1 = 4$ and $\beta_2 = 25$. Unlike hard clamping of gate pre-activations, the smooth cap preserves nonzero gradients away from saturation boundaries, which we find to give better training behavior.

C Derivation of Quantile Balancing

This appendix derives the Quantile Balancing (QB) updates used in §2.3 from optimal balanced assignment, following [111]; the assignment perspective on expert load balancing goes back to BASE Layers [67] and BIP [116]. Let $\boldsymbol{s} \in \mathbb{R}^{m \times n}$ collect the router scores of m tokens over n experts, where each token selects exactly k experts and $x_{i,j} \in \{0,1\}$ indicates whether token i is assigned to expert j. The maximum-score balanced assignment, in which each expert serves exactly mk/n tokens (assumed integral), is

$$\max_{x_{i,j} \in \{0,1\}} \sum_{i,j} x_{i,j} s_{i,j} \qquad \text{s.t.} \qquad \sum_j x_{i,j} = k, \qquad \sum_i x_{i,j} = \frac{mk}{n}. \tag{20}$$

Linear relaxation and duality Relaxing $x_{i,j} \in \{0,1\}$ to $x_{i,j} \in [0,1]$ turns Eq. 20 into a linear program, whose optimum is integral by the standard integrality of the bipartite b-matching polytope; the relaxation is therefore exact. Introducing free multipliers α_i and β_j for the token- and expert-side equality constraints, respectively, the relaxed problem can be written in max–min form as

$$\max_{x_{i,j} \in [0,1]} \min_{\alpha_i, \beta_j} \sum_{i,j} x_{i,j} s_{i,j} - \sum_i \alpha_i \Big(\sum_j x_{i,j} - k \Big) - \sum_j \beta_j \Big(\sum_i x_{i,j} - \tfrac{mk}{n} \Big). \tag{21}$$

The objective is linear in each of $\boldsymbol{x}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$, and the feasible sets are convex, so the minimax theorem allows exchanging the order of optimization:

$$\min_{\alpha_i, \beta_j} \max_{x_{i,j} \in [0,1]} \sum_{i,j} x_{i,j} \big(s_{i,j} - \alpha_i - \beta_j \big) + k \sum_i \alpha_i + \frac{mk}{n} \sum_j \beta_j. \tag{22}$$

The inner maximum is separable over entries, with $x^*_{i,j} = 1$ if $s_{i,j} - \alpha_i - \beta_j > 0$ and $x^*_{i,j} = 0$ if $s_{i,j} - \alpha_i - \beta_j < 0$; the tie case has measure zero in practice. Substituting x^* gives the convex dual objective

$$\min_{\alpha_i, \beta_j} \; \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) := \sum_{i,j} \max\big(0,\; s_{i,j} - \alpha_i - \beta_j\big) + k \sum_i \alpha_i + \frac{mk}{n} \sum_j \beta_j. \tag{23}$$

Algorithm 1: The alternating QB solver.

Input: score matrix $\boldsymbol{s} \in \mathbb{R}^{m\times n}$
Output: assignment $\boldsymbol{x} \in \{0,1\}^{m\times n}$
Initialize $\boldsymbol{\beta} = \mathbf{0}_{1\times n}$;
for $t = 1, 2, \cdots, T$ **do**
  $\boldsymbol{\alpha} \leftarrow \text{desc\_sort}(\boldsymbol{s} - \boldsymbol{\beta},\ \text{axis=1})_{[:,\,k:k+1]}$
  $\boldsymbol{\beta} \leftarrow \text{desc\_sort}(\boldsymbol{s} - \boldsymbol{\alpha},\ \text{axis=0})_{[mk/n:\,mk/n+1]}$
end
return $\boldsymbol{x}$ *with* $x_{i,j} = 1$ if $j \in \text{argtop}_k(\boldsymbol{s}_i - \boldsymbol{\beta})$, and 0 otherwise

Exact coordinate minimization We minimize Eq. 23 by alternately solving for $\boldsymbol{\alpha}$ with $\boldsymbol{\beta}$ fixed and vice versa; each subproblem admits a closed-form exact solution. With $\boldsymbol{\beta}$ fixed, the problem decouples over tokens, and for token i we solve

$$\min_{\alpha}\ k\alpha + \sum_{j} \max\big(0,\ s_{i,j} - \beta_j - \alpha\big). \tag{24}$$

This objective is piecewise linear in α with slope k minus the number of margins $s_{i,j} - \beta_j$ exceeding α; it is therefore minimized exactly when k margins lie above α, i.e., for any α_i^* between the k-th and $(k{+}1)$-th largest entries of $\boldsymbol{s}_i - \boldsymbol{\beta}$. By convention we take the $(k{+}1)$-th largest entry, which is equivalently the $(1 - k/n)$-th quantile:

$$\alpha_i^* = \text{quantile}_{1-k/n}\big(\boldsymbol{s}_i - \boldsymbol{\beta}\big). \tag{25}$$

Symmetrically, with $\boldsymbol{\alpha}$ fixed, expert j solves $\min_{\beta} \frac{mk}{n}\beta + \sum_i \max(0, s_{i,j} - \alpha_i - \beta)$, whose minimizer is the $(mk/n{+}1)$-th largest entry of $\boldsymbol{s}_{:,j} - \boldsymbol{\alpha}$, again the $(1 - k/n)$-th quantile:

$$\beta_j^* = \text{quantile}_{1-k/n}\big(\boldsymbol{s}_{:,j} - \boldsymbol{\alpha}\big). \tag{26}$$

Both updates are thus the same quantile along the token and expert axes, respectively, which gives the method its name. Fig. 5 illustrates the expert-side update as equalizing the accepted upper tail of each expert's margin distribution, and Alg. 1 summarizes the resulting alternating solver.

From assignment to routing At the optimum of Eq. 23, $x_{i,j}^* = 1$ if and only if $s_{i,j} - \alpha_i^* - \beta_j^* > 0$; combined with the token constraint $\sum_j x_{i,j}^* = k$, the selected experts are exactly the Top-k entries of $\boldsymbol{s}_i - \boldsymbol{\beta}^*$. Routing therefore requires only the expert thresholds $\boldsymbol{\beta} \in \mathbb{R}^n$ (equivalently, the bias $\boldsymbol{b} = -\boldsymbol{\beta}$ of Eq. 13), while the token thresholds $\boldsymbol{\alpha} \in \mathbb{R}^m$ are intermediate variables tied to the dynamic training batch and are discarded. This asymmetry preserves train–inference consistency: at deployment, routing is a fixed Top-k selection with a frozen bias, and no quantile computation is needed.

Relation to sign-based loss-free updates The expert-side subproblem underlying Eq. 26 has (sub)gradient

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \frac{mk}{n} - \sum_{i=1}^{m} \chi\big(s_{i,j} - \alpha_i - \beta_j > 0\big), \tag{27}$$

i.e., the target load minus the observed load of the expert j. A SignSGD step on this objective recovers the fixed-step sign update of auxiliary-loss-free balancing [30], up to the sign convention $\boldsymbol{b} = -\boldsymbol{\beta}$: the sign update retains only the direction of the load error in Eq. 27, whereas QB jumps directly to the exact coordinate minimizer of the same dual objective. This view explains both why QB requires no learning-rate-like hyperparameter and why it equilibrates within a few update steps even for nearly 10^3 experts. QB is likewise related to BIP [116], which solves the same assignment with inequality constraints $\sum_j x_{i,j} \le k$ and $\sum_i x_{i,j} \le mk/n$; the induced non-negativity constraints on $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ add a $\max(0, \cdot)$ clipping to both updates, which can only suppress over-selected experts without promoting under-selected ones, and markedly slows equilibration in our experiments. Finally, the resulting fixed-Top-k routing is related to expert-specific threshold routing but differs from Expert Threshold routing, which maintains EMA thresholds and permits a variable number of selected experts per token [112].

D Histogram-Based Quantile Estimation

The QB update of Eq. 14 asks for a quantile taken over the whole training step: for each of the n experts, the $(1-k/n)$-th quantile of the margins $s_{i,j} - \alpha_i$, where the token count m spans millions of tokens sharded across data-parallel ranks

and gradient-accumulation steps. Gathering $O(mn)$ margins for an exact quantile is impractical inside the training loop. The key observation is that the update never needs the margins themselves, only their per-expert distribution, which a histogram summarizes at fixed cost. Kimi K3 therefore maintains a binned histogram per expert and reads the quantile from it. Concretely, we histogram the *required bias* $r_{i,j} := \alpha_i - s_{i,j}$, the bias that would place expert j exactly at token i's cutoff; negating the margins reverses their order, so the QB target $\widehat{b}_j$ of Eq. 14 is exactly the (k/n)-quantile of $r_{:,j}$.

Binning range The first question is which interval to bin over, and here the required bias helps: its range is bounded by the current bias itself. Router scores are sigmoid outputs, so $s_{i,j} \in (0, 1)$, and the cutoff α_i is itself the biased score $s_{i,j'} + b_{j'}$ of some expert j', so it lies in $(b_{\min},\ 1 + b_{\max})$, with $b_{\min}$ and $b_{\max}$ the extremes of the current bias. Every $r_{i,j}$ therefore falls in $[b_{\min} - 1,\ b_{\max} + 1]$. We partition this interval into B uniform bins, which we find sufficient in practice, and recompute the range every step, so the bin width $w = (b_{\max} - b_{\min} + 2)/B$ stays adapted to the bias as it spreads to correct imbalance.

Accumulation and recovery The rest of the procedure follows the structure of a training step. During each forward pass, every rank scatter-adds its local $r_{i,j}$ values into a per-expert count matrix $\mathbf{H} \in \mathbb{N}^{n \times B}$, accumulating over all micro-batches with no communication. At the end of the step, a single all-reduce sums the local counts into the global histogram, and every rank recovers the quantile from the same pooled counts. Each expert's histogram counts every token once, so the target rank is exactly the target load $q = mk/n$ of § 2.3.3, now taken over the full step: we select the first bin whose cumulative count reaches $\lceil q \rceil$ and interpolate linearly within it. If bin β_j is selected, with cumulative count c_j before it and h_j counts inside it, then

$$\widehat{b}_j = b_{\min} - 1 + \Big(\beta_j + \mathrm{clip}\big(\tfrac{q - c_j}{h_j},\ 0,\ 1\big)\Big) w,$$

and the resulting biases are mean-centered as in Eq. 14.

Properties Three properties make this estimator practical at scale. First, it is accurate: the cumulative counts are exact at bin edges, so the true quantile and its estimate lie in the same bin and the error is bounded by the bin width w; with $B = 1000$ this is at most a few 10^{-3}, and we observe no measurable residual load imbalance. Second, it is cheap: the only communication is one integer all-reduce of nB values per layer per step, independent of m, which in our configuration is below 1% of the cost of exchanging the raw margins over a process group every micro-batch, the natural alternative. Third, it estimates the right quantity: because counts are additive, the global histogram is exactly invariant to how tokens are partitioned across ranks or accumulation steps, and the estimate is the quantile of the pooled global batch rather than an average of per-rank quantiles, which generally differs. As a further refinement, maintaining an exponential moving average of the estimated quantiles across steps reduces batch-to-batch sampling noise and can improve load balance still further.

E MoonEP General Upper Bound Proof

Let $m_r(P)$ denote the number of redundant experts placed on rank r under plan P. For a router output I, the planning objective is to minimize the maximum number of redundant experts on any rank, i.e., $M(I) = \min_P \max_r \{m_r(P)\}$. We prove that $M(I) \le E/R$ always holds (Theorem 1) and that this bound is essentially tight: there exist router outputs for which $M = \lceil E(R-1)/R^2 \rceil \approx E/R$ (Theorem 2).

Proof of Theorem 1 (General Upper Bound) The goal is to prove that $M(I) \le E/R$ holds for any router output I. Key lemma: there exists a plan P^* such that every EP rank receives exactly the same number of tokens ($S \times K$), and the remote tokens of each rank come from only one other EP rank. The construction is as follows: initially, every rank holds only local tokens, and ranks are classified as underloaded or overloaded accordingly. We repeatedly pick an underloaded rank and an overloaded rank, and migrate tokens from the overloaded rank to fill the underloaded rank exactly up to the balanced value $S \times K$; the overloaded rank may remain overloaded, become exactly balanced, or become underloaded, and is put back into the corresponding set. This is repeated until all ranks are perfectly balanced. Each fill makes one underloaded rank balanced and it never changes afterwards, so the process terminates after at most $R - 1$ fills; meanwhile, each rank is filled at most once, so its remote tokens come from a single rank, which proves the lemma. Consequently, supposing all remote tokens of rank r come from rank s; these tokens belong to at most E/R local experts on rank s, hence $m_r(P^*) \le E/R$, and therefore

$$M(I) = \min_P \max_r \{m_r(P)\} \le \max_r \{m_r(P^*)\} \le \frac{E}{R} \tag{28}$$

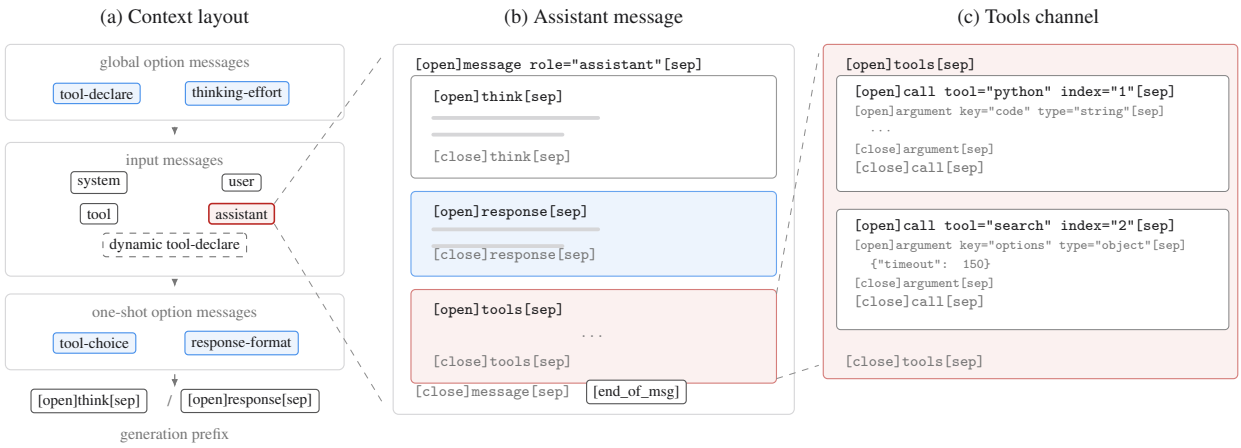


Figure 16: Structure of the Kimi K3 chat template. **(a)** Context layout: global option messages precede the input messages, while one-shot option messages follow them, so that per-request options leave the history KV cache intact; dynamically loaded tools are injected mid-session as input option messages (dashed). **(b)** Anatomy of an assistant message: the body is organized into `think`, `response`, and `tools` channels. **(c)** Expansion of the `tools` channel: parallel tool calls are indexed so that tool results can be matched to their calls, and arguments are typed.

Proof of Theorem 2 (Tightness of the Upper Bound) Construct a router output I^* as follows: the experts on EP rank 0 receive no tokens, while all experts on the other $R-1$ ranks share all tokens evenly. Then all $S \times K \times R$ tokens are evenly divided among $E(R-1)/R$ experts, so each expert receives $\frac{SKR^2}{E(R-1)}$ tokens. Under any plan P, rank 0 must receive $S \times K$ tokens, all of which are remote, and these tokens involve at least $SK \Big/ \frac{SKR^2}{E(R-1)} = \frac{E(R-1)}{R^2}$ distinct experts; taking the ceiling, rank 0 requires at least $\left\lceil \frac{E(R-1)}{R^2} \right\rceil$ redundant experts, hence $M(I^*) \geq \left\lceil \frac{E(R-1)}{R^2} \right\rceil$. Conversely, by constructing a plan with the filling procedure from the proof of Theorem 1 and migrating tokens expert-wise preferentially, the number of redundant experts on every rank can be kept within this value, so equality holds. Since $\left\lceil \frac{E(R-1)}{R^2} \right\rceil \approx \frac{E}{R}$ when R is large, the upper bound in Theorem 1 is essentially tight: there is no general upper bound significantly smaller than E/R.

F Chat Template

The Kimi K3 chat template is redesigned around three goals. The first is *extensibility*: new capabilities should be introduced through backward-compatible message formats rather than template revisions, so that a single template serves the entire model generation. The second is a *low alignment tax*: the format should be learnable with minimal supervised data, supporting a pipeline in which a lightly fine-tuned pre-trained model can proceed directly to reinforcement learning. The third is *decoding friendliness*: the structure should admit simple encoders, streaming parsers, and grammar-constrained enforcers. To these ends, the template adopts XTML (eXtensible Token Markup Language), an XML-like markup in which the angle-bracket syntax is replaced by three reserved special tokens: `[open]`, `[sep]` and `[close]`, with an additional `[end_of_msg]` token as the generation stop marker. An element `[open]tag attr="value"[sep] ... [close]tag[sep]` is isomorphic to its XML counterpart, but every structural boundary is an explicit special token, which removes tokenization ambiguity at element boundaries and simplifies constrained decoding.

Messages and zones The top-level unit of the context is the message, and messages fall into two categories by origin (Fig. 16a). *Input messages* serialize the `messages` field of the request, covering the familiar system, user, assistant, and tool roles. *Option messages* translate request options into instructions that the model reads in context, and their placement reflects their scope. *Global options*—the tool declaration (`type="tool-declare"`) and the reasoning-effort setting—appear before all input messages: they govern the whole session and rarely change, so modifying them invalidates the KV cache anyway. *One-shot options* (`tool_choice`, `response_format`) are appended after the input messages, so that per-request changes leave the history KV cache intact. A third kind, the *input option message*, is interleaved with input messages to supplement or override a global option mid-session. This mechanism

supports *dynamically loaded tools*: tools retrieved or loaded during a conversation are announced through an additional tool-declare message, after which the model's available toolset expands without rebuilding the preceding context.

Channels The body of an assistant message is organized into *channels*, a concept inspired by OpenAI's Harmony response format [85]: `think` carries the reasoning trace, `response` the user-visible answer, and `tools` the tool calls (Fig. 16b). The two generation modes are selected purely through the generation prefix—`[open]think[sep]` for thinking mode and `[open]response[sep]` for instruct mode—rather than through separate templates. Kimi K3 supports only *preserved thinking*: in thinking mode, the think channel is always retained in the history—kept even when its content is empty—so that the model observes a consistent message structure across turns; in instruct mode, historical messages contain only the response and tools channels.

Tool calling Within the tools channel, each call carries `tool` and `index` attributes; the index numbers parallel calls within a message, and each tool-result message repeats the same `tool`/`index` pair and follows the order of its call, so that results are unambiguously associated with calls. Arguments are typed: string arguments appear as raw text, while values of other JSON types are compactly serialized. Free-form text such as code is therefore a first-class citizen rather than an escaped JSON string. A pure-JSON fallback block covers inputs whose arguments cannot be decomposed into typed argument blocks; it occurs only in input tokens, never in model outputs, and its loss is masked during training.

Reasoning effort and options Reasoning effort is exposed as a global option message of type `thinking-effort`, inserted after the tool declaration and before the input messages. Instead of modifying the generation prefix or exposing a token budget, the message states the requested level in natural language and acts as a generation-constraint instruction. The schema reserves four levels (`low`, `medium`, `high`, and `max`), of which Kimi K3 supports a subset. This representation decouples the effort interface from the template syntax, and it aligns directly with the effort-conditioned training described in §4.1.1 and §4.1.2.

More broadly, this is the common implementation of all option messages: `tool_choice`, `response_format`, and `thinking-effort` are each translated into a short natural-language instruction placed in context, rather than into dedicated special syntax. Because the pre-trained model already follows such instructions well, new options can be introduced with little or no additional training—a direct embodiment of the low-alignment-tax design principle stated above.