Kimi 线性：一种表达性强、高效的注意力架构

Technical Report of Kimi Linear

Kimi Team
https://github.com/MoonshotAI/Kimi-Linear

Abstract

我们引入了 Kimi Linear，这是一种混合线性注意力架构，在各种场景下的公平比较中首次超越了完整注意力机制的表现，包括短上下文、长上下文以及强化学习（RL）缩放场景。其核心是 Kimi Delta 注意力（KDA），一种表达能力强的线性注意力模块，它在 Gated DeltaNet [**yang-2025-gdn**] 的基础上引入了更细粒度的门控机制，从而更有效地利用有限状态循环神经网络（RNN）的内存。我们专门设计的分块算法通过一种特殊的对角加低秩（DPLR）转移矩阵变体实现了高硬件效率，相较于通用的 DPLR 公式显著减少了计算量，同时仍与经典的差分规则保持更高的一致性。

我们基于逐层混合的 KDA 与多头潜在注意力（MLA）架构，对一个具有 30 亿激活参数和 480 亿总参数的 Kimi 线性模型进行了预训练。我们的实验表明，在相同的训练方案下，Kimi 线性模型在所有评估任务中均显著优于完整的 MLA 模型，同时将 KV 缓存使用量减少了高达 75%，并在 100 万上下文长度下实现了高达 6× 的解码吞吐量。这些结果表明，Kimi 线性模型可作为完整注意力架构的即插即用替代方案，在性能和效率方面表现更优，尤其适用于输入和输出长度较长的任务。

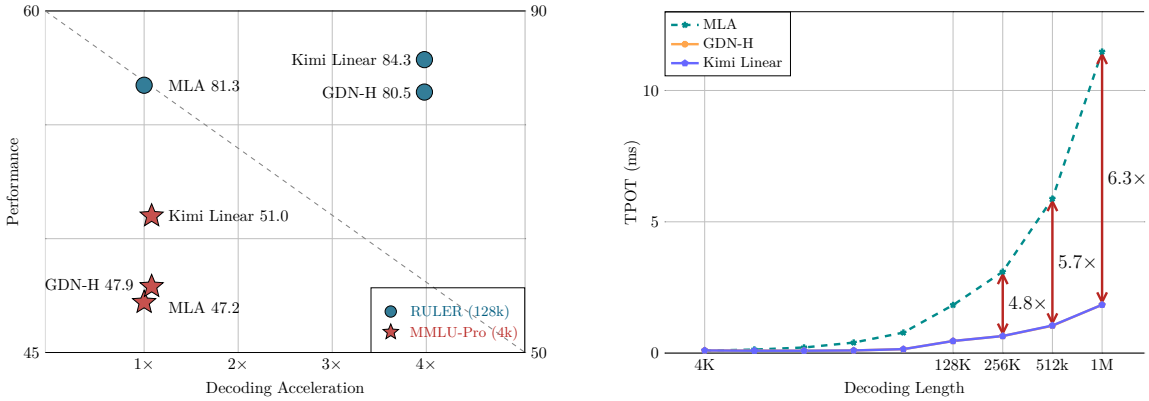
为了支持进一步的研究，我们开源了 KDA 内核和 vLLM 实现 [1]，并发布预训练和指令微调的模型检查点。[2]

1 引言

随着大规模语言模型（LLMs）不断演进为日益强大的智能体 [**kimi2025k2**]，推理过程中的计算需求——尤其是在长时程和强化学习（RL）情景下——正逐渐成为核心瓶颈。这种向 *RL 推理时缩放* [**kimiteam2025kimik15scalingreinforcement**, **guo2025deepseek**, **qu2025survey**, **plaat2024reasoning**, **lai2025survey**] 的转变，要求模型在推理阶段处理更长的轨迹、工具使用交互以及复杂的决策空间，暴露出标准注意力机制的根本性低效问题。特别是，Softmax 注意力机制带来的二次时间复杂度以及线性增长的键-值（KV）缓存，引入了显著的计算与内存开销，制约了吞吐量、上下文长度的缩放能力以及实时交互性能。

线性注意力 [**katharopoulos-2020-transformers**] 提供了一种降低计算复杂度的合理方法，但由于表达能力有限，历史上在语言模型化任务中表现不佳——即使对于短序列也是如此。最近的进展通过两项创新显著缩小了这一差距：门控或衰减机制 [**sun-2023-retnet**, **mamba2**, **yang-etal-2024-gla**] 以及 delta

[1] https://github.com/fla-org/flash-linear-attention/tree/main/fla/ops/kda
[2] https://huggingface.co/moonshotai/Kimi-Linear-48B-A3B-Instruct



(a) (b)

图 1: (a) 性能 vs. 加速。在与 1.4 万亿训练 token 进行严格公平比较时，在 MMLU-Pro（4k 上下文长度，红色星号）上，Kimi 线性的（Linear）性能（51.0）领先，且速度相近。在 RULER（128k 上下文长度，蓝色圆圈）上，该模型达到帕累托最优，实现最高性能（84.3）和 3.98× 加速。(b) 每个输出 token 的时间（TPOT）vs. 解码长度。Kimi 线性的（蓝色线）保持低 TPOT，与 GDN-H 相当，且在长序列上优于 MLA。这支持更大的批量，实现了 6.3× 更快的 TPOT（1.84ms vs. 11.48ms），优于 MLA 在 100 万 token 时的表现。

规则 [**schlag-2021-deltanet**, **yang-2024-parallelizing**, **yang-2025-gdn**, **peng-2025-rwkv7**]。这些发展共同推动了线性注意力在中等长度序列上的性能接近 Softmax 注意力水平。然而，纯粹的线性结构仍然受到有限状态容量的根本限制，使得长序列建模和上下文内检索在理论上具有挑战性 [**wen2024rnns**, **arora2024simple**, **jelassi-2024-repeat**]。

结合 Softmax 与线性注意力的混合架构——在主要采用更快的线性层的同时，辅以少量全局注意力层——因此成为质量与效率之间的一种实际折衷方案 [**lieber2024jamba**, **mamba2hybrid**, **minimax2025minimax01**, **blakeman2025nemotron**, **gu2025jetnemotronefficientlanguagemodel**, **qwen3next2025**]。然而，以往的混合模型通常仅在有限规模下运行，或缺乏在多样化基准上的全面评估。核心挑战依然存在：开发一种注意力架构，在保持与全注意力相当或更优的质量的同时，实现显著的速度和内存效率提升——这是推动下一代基于智能体、解码密集型大模型发展的关键一步。

在本工作中，我们提出了 **Kimi Linear**，这是一种混合线性注意力架构，旨在满足智能体智能和测试时缩放的效率需求，同时不牺牲质量。其核心是 **Kimi Delta Attention (KDA)**，一种硬件高效的线性注意力模块，它在 Gated DeltaNet [**yang-2025-gdn**] 的基础上引入了更细粒度的门控机制。与类似 Mamba2 [**mamba2**] 的 GDN 采用粗粒度的头级遗忘门不同，KDA 引入了通道级变体，其中每个特征维度独立维持一个遗忘率，类似于 Gated Linear Attention (GLA) [**yang-etal-2024-gla**]。这种细粒度设计使得对有限状态 RNN 记忆的调控更加精确，从而释放了混合架构中 RNN 风格模型的潜力。

至关重要的是，KDA 使用一种特殊的对角加低秩 (DPLR) 矩阵 [**gu-2022-efficiently**, **peng-2025-rwkv7**] 来参数化其转移动态，从而实现一种定制化的分块并行算法，该算法相较于一般的 DPLR 表达式显著降低了计算量，同时仍与经典的 delta 规则保持一致。

Kimi Linear 以统一的 3:1 比例在线性交错结构中交替使用 KDA 和周期性的全注意力层。这种混合结构在长序列生成过程中将内存和 KV 缓存使用量最多降低 75%，同时通过全注意力层保持全局信息流动。通过匹配尺度的预训练和评估，我们表明，Kimi Linear 在短上下文、长上下文以及强化学习风格的后训练任务中，始终与强大的全注意力基准模型相当或表现更优——并且在 1M 上下文长度下，解码吞吐量最高提升 6×。

为促进进一步研究，我们发布了集成 vLLM 的开源 KDA 内核，以及预训练和指令微调的检查点。这些组件可直接用于现有的全注意力流水线，无需修改缓存或调度接口，从而便于混合架构的研究。

贡献

- **Kimi Delta Attention (KDA):** 一种线性注意力机制，通过改进的循环记忆管理和硬件效率，优化了门控差分规则。
- **Kimi 线性架构：** 一种混合设计，采用 3:1 的 KDA 到全局注意力比例，在降低内存占用的同时超越了全注意力的质量。
- **大规模公平的实证验证：** 通过 1.4T token 的训练运行，Kimi Linear 在短/长上下文以及强化学习风格的评估中均优于完整注意力机制和其他基准方法，同时开源了全部内核、集成 vLLM，并发布了检查点。

2 初步的

在本节中，我们介绍与我们提出的 Kimi Delta Attention 相关的技术背景。

2.1 符号标记

在本文中，我们定义 $\square_t \in \mathbb{R}^{d_k}$ 或 $\mathbb{R}^{d_v}$ ，s.t.,$\square \in \{\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}, \boldsymbol{o}, \boldsymbol{u}, \boldsymbol{w}\}$ 表示第 t 个对应的列向量，$\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$ 表示矩阵形式的状态。$\mathbf{M}$ 和 $\mathbf{M}^-$ 分别表示带和不带对角线元素的下三角掩码；为方便起见，我们也将其写作 Tril 和 StrictTril 。

分块公式化 假设该序列被分割成 L/C 个块，每个块的长度为 C 。我们定义 $\square_{[t]} \in \mathbb{R}^{C \times d}$ ，其中 $\square \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{U}, \mathbf{W}\}$ 为矩阵，用于堆叠第 t 个块内的向量，$\square^r_{[t]} = \square_{tC+r}$ 为该块的第 r 个元素。注意 $t \in [0, L/C), r \in [1, C]$ 。状态矩阵也重新编号，使得 $\boldsymbol{S}^i_{[t]} = \boldsymbol{S}_{tC+i}$ 。此外，$\boldsymbol{S}_{[t]} := \boldsymbol{S}^0_{[t]} = \boldsymbol{S}^C_{[t-1]}$ ，即一个块的初始状态是前一个块的最后一个状态。

衰变模型 我们定义累积衰减 $\gamma^{i \to j}_{[t]} := \prod_{k=i}^{j} \alpha^k_{[t]}$ ，并将 $\gamma^{1 \to r}_{[t]}$ 简写为 $\gamma^r_{[t]}$ 。此外，$\mathcal{A}_{[t]} := \mathcal{A}^{i/j}_{[t]} \in \mathbb{R}^{C \times C}$ 是元素为 $\gamma^i_{[t]}/\gamma^j_{[t]}$ 的矩阵。$\mathrm{Diag}\left(\boldsymbol{\alpha}_t\right)$ 表示细粒度衰减，$\mathrm{Diag}\left(\boldsymbol{\gamma}^{i \to j}_{[t]}\right) := \prod_{k=i}^{j} \mathrm{Diag}\left(\boldsymbol{\alpha}^k_{[t]}\right)$ ，而 $\boldsymbol{\Gamma}^{i \to j}_{[t]} \in \mathbb{R}^{C \times d_k}$ 是从 $\boldsymbol{\gamma}^i_{[t]}$ 到 $\boldsymbol{\gamma}^j_{[t]}$ 的矩阵堆叠。

2.2 线性注意力与门控差分规则

线性注意力作为在线学习。 线性注意力 [**katharopoulos-2020-transformers**] 维持一个矩阵值的循环状态，用于累积键-值关联：

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top, \qquad \boldsymbol{o}_t = \mathbf{S}_t^\top \boldsymbol{q}_t.$$

从快速权重的角度来看 [**schlag-2021-deltanet**, **schlag-2021-learning**]，$\mathbf{S}_t$ 作为一种联想记忆，存储了从键到值的瞬时映射。该更新可被视为在无界相关系数目标上执行梯度 下降

$$\mathcal{L}_t(\mathbf{S}) = -\langle \mathbf{S}^\top \boldsymbol{k}_t, \boldsymbol{v}_t \rangle,$$

这会持续强化最近的关键-值对，而不会遗忘。然而，这种目标并未提供删除哪些记忆的准则，累积状态无限制增长，导致长上下文中的干扰。

DeltaNet：基于重构损失的在线梯度下降。 DeltaNet [**schlag-2021-deltanet**] 将此循环重新解释为在重构目标上的在线梯度 下降：

$$\mathcal{L}_t(\mathbf{S}) = \tfrac{1}{2} \|\mathbf{S}^\top \boldsymbol{k}_t - \boldsymbol{v}_t\|^2.$$

使用学习率 β_t 进行一次梯度步进得到

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \nabla_{\mathbf{S}} \mathcal{L}_t(\mathbf{S}_{t-1}) = (\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top)\mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top.$$

该规则——经典的 *delta* 规则——将 $\mathbf{S}$ 视为一种可学习的联想记忆，能够持续修正自身以逼近映射 $\boldsymbol{k}_t \mapsto \boldsymbol{v}_t$。秩-1 更新结构等价于广义的 Householder 变换，支持硬件高效的分块并行化 [**bischof-wy-1987**, **yang-2024-parallelizing**]。

门控 DeltaNet 作为权重衰减。 尽管 DeltaNet 稳定了学习过程，但它仍然会无限期地保留过时的关联。门控 DeltaNet（GDN） [**yang-2025-gdn**] 引入了一个标量遗忘门 $\alpha_t \in [0,1]$ ，从而实现

$$\mathbf{S}_t = \alpha_t(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top)\mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top.$$

此处，α_t 作为快速权重 [**behrouz2025atlas**] 的一种形式的 权重衰减，实现了类似于数据依赖性 L_2 正则化的遗忘机制。这一简单而有效的改进提供了一种合理的方法来控制记忆寿命并缓解干扰，从而在保持 DeltaNet 并行结构的同时，提升了稳定性与长上下文泛化能力。

从这一视角来看，我们观察到 GDN 可以被解释为一种乘法形式的位置编码，其中状态转移矩阵是数据相关的且可学习的，从而放松了 RoPE [**yang2025path**] 的正交性约束。[3]

3 Kimi Delta 注意力：通过细粒度门控改进 Delta 规则

我们提出 Kimi Delta Attention (KDA)，这是一种新的门控线性注意力变体，通过引入细粒度的对角化门 $\mathrm{Diag}(\boldsymbol{\alpha}_t)$ 来优化 GDN 的标量衰减机制，从而实现对记忆衰减和位置感知的精细控制（如 §6.1 所讨论）。我们首先介绍 KDA 的分块并行化方法，展示一系列秩-1 矩阵变换如何被压缩为稠密表示，同时在对角门控下保持稳定性。随后，我们强调 KDA 相较于标准 DPLR (对角加低秩) 公式 [**gu-2022-efficiently**, **peng-2025-rwkv7**] 的效率优势。

$$\mathbf{S}_t = \left(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) \mathrm{Diag}\left(\boldsymbol{\alpha}_t\right) \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top \in \mathbb{R}^{d_k \times d_v}; \qquad \boldsymbol{o}_t = \mathbf{S}_t^\top \boldsymbol{q}_t \in \mathbb{R}^{d_v} \tag{1}$$

3.1 硬件高效的分块算法

通过将公式 1 的循环部分展开为分块形式，我们得到：

$$\mathbf{S}_{[t]}^r = \underbrace{\left(\prod_{i=1}^{t}\left(\mathbf{I} - \beta_{[t]}^i \boldsymbol{k}_{[t]}^i \boldsymbol{k}_{[t]}^{i\top}\right)\mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^i)\right)}_{:=\mathbf{P}_{[t]}^r} \cdot \mathbf{S}_{[t]}^0 + \underbrace{\sum_{i=1}^{r}\left(\prod_{j=i}^{t}\left(\mathbf{I} - \beta_{[t]}^j \boldsymbol{k}_{[t]}^j \boldsymbol{k}_{[t]}^{j\top}\right)\mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^j)\right) \cdot \beta_{[t]}^i \boldsymbol{k}_{[t]}^i \boldsymbol{v}_{[t]}^{i\top}}_{:=\mathbf{H}_{[t]}^r} \tag{2}$$

WY 表示 通常采用 [**bischof-wy-1987**] 将一系列秩-1 更新打包成一个紧凑的表示。我们遵循 Comba [**hu2025comba**] 中的公式 $\mathbf{P}$，以减少后续计算中对额外逆矩阵的需求。

$$\mathbf{P}_{[t]}^r = \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r} \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^{i\to r}) \boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} \qquad \mathbf{H}_{[t]}^r = \sum_{i=1}^{t} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} \tag{3}$$

[3] 当状态变换矩阵保持正交性时，绝对位置编码也可独立应用于 $\boldsymbol{q}$ 和 $\boldsymbol{k}$ ，在注意力计算 [**kexuefm-11033**] 过程中转换为相对位置编码。

其中辅助向量 $\boldsymbol{w}_t \in \mathbb{R}^{d_k}$ 和 $\boldsymbol{u}_t \in \mathbb{R}^{d_v}$ 通过以下循环关系计算得到：

$$\boldsymbol{w}_{[t]}^r = \beta_{[t]}^r \left(\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) \boldsymbol{k}_{[t]}^r - \sum_{i=1}^{r-1} \boldsymbol{w}_{[t]}^i \left(\boldsymbol{k}_{[t]}^{i\top} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r} \right) \boldsymbol{k}_{[t]}^r \right) \right) \tag{4}$$

$$\boldsymbol{u}_{[t]}^r = \beta_{[t]}^r \left(\boldsymbol{v}_{[t]}^r - \sum_{i=1}^{r-1} \boldsymbol{u}_{[t]}^i \left(\boldsymbol{k}_{[t]}^{i\top} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r} \right) \boldsymbol{k}_{[t]}^r \right) \right) \tag{5}$$

UT 变换。 我们应用 UT 变换 [**joffrain-2006-ut**] 来减少非 matmul 的浮点运算次数，这对于在训练期间实现更好的硬件利用率至关重要。

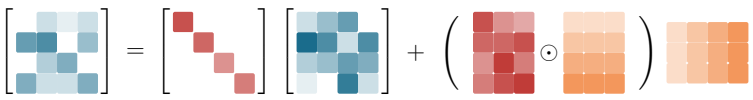
$$\mathbf{M}_{[t]} = \left(\mathbf{I} + \mathrm{StrictTril} \left(\mathrm{Diag}\left(\beta_{[t]}\right) \left(\boldsymbol{\Gamma}_{[t]}^{1\to C} \odot \mathbf{K}_{[t]} \right) \left(\frac{\mathbf{K}_{[t]}}{\boldsymbol{\Gamma}_{[t]}^{1\to C}} \right)^\top \right) \right)^{-1} \mathrm{Diag}\left(\beta_{[t]}\right) \tag{6}$$

$$\mathbf{W}_{[t]} = \mathbf{M}_{[t]} \left(\boldsymbol{\Gamma}_{[t]}^{1\to C} \odot \mathbf{K}_{[t]} \right), \qquad \mathbf{U}_{[t]} = \mathbf{M}_{[t]} \mathbf{V}_{[t]} \tag{7}$$

下三角矩阵的逆矩阵可以通过高斯消元中的前向置换，以迭代按行的方式高效计算 [**grcar__2011**].

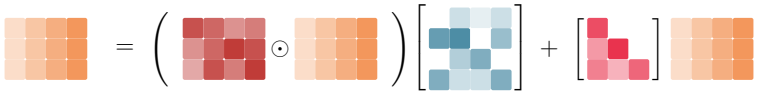
等价地，以矩阵形式，我们可以分块地更新状态：

$$\mathbf{S}_{[t+1]} = \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^C) \mathbf{S}_{[t]} + \left(\boldsymbol{\Gamma}_{[t]}^{i\to C} \odot \mathbf{K}_{[t]} \right)^\top \left(\mathbf{U}_{[t]} - \mathbf{W}_{[t]} \mathbf{S}_{[t]} \right) \in \mathbb{R}^{d_k \times d_v} \tag{8}$$



在输出阶段，我们采用块间循环、块内并行的策略，以最大化矩阵乘法吞吐量，从而充分挖掘张量核心的计算潜力。

$$\mathbf{O}_{[t]} = \underbrace{\left(\boldsymbol{\Gamma}_{[t]}^{1\to C} \odot \mathbf{Q}_{[t]} \right) \mathbf{S}_{[t]}}_{\text{inter chunk}} + \underbrace{\mathrm{Tril}\left(\left(\boldsymbol{\Gamma}_{[t]}^{1\to C} \odot \mathbf{Q}_{[t]} \right) \left(\frac{\mathbf{K}_{[t]}}{\boldsymbol{\Gamma}_{[t]}^{1\to C}} \right)^\top \right)}_{\text{intra chunk}} \underbrace{\left(\mathbf{U}_{[t]} - \mathbf{W}_{[t]} \mathbf{S}_{[t]} \right)}_{\text{“pseudo”-value term}} \in \mathbb{R}^{C \times d_v} \tag{9}$$



3.2 效率分析

在表示容量方面，KDA 与广义的 DPLR 公式一致，即 $\mathbf{S}_t = (\mathbf{D} - \boldsymbol{a}_t \boldsymbol{b}_t^\top)\mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$，二者均表现出细粒度衰减行为。然而，这种细粒度衰减在除法运算过程中引入了数值精度问题（例如，公式 Eq. 9 中的块内计算）。为解决此问题，先前的工作如 GLA [**yang-etal-2024-gla**] 在对数域中进行计算，并在全精度下引入二级分块。然而，这种方法阻碍了半精度矩阵乘法的充分利用，显著降低了算子速度。

通过将变量 $\boldsymbol{a}$ 与 $\boldsymbol{b}$ 均绑定至 $\boldsymbol{k}$，KDA 有效缓解了这一瓶颈——将二级分块矩阵计算次数从四次减少至两次，并进一步消除了三次额外的矩阵乘法。因此，与 DPLR 公式相比，KDA 的算子效率提升了约 100%。详细分析见 §6.2。

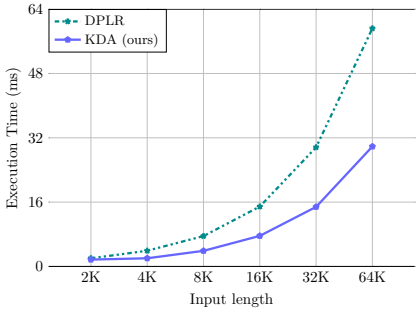


图 2: 不同输入长度下内核的执行时间，批量大小统一为 1，注意力头数为 16。

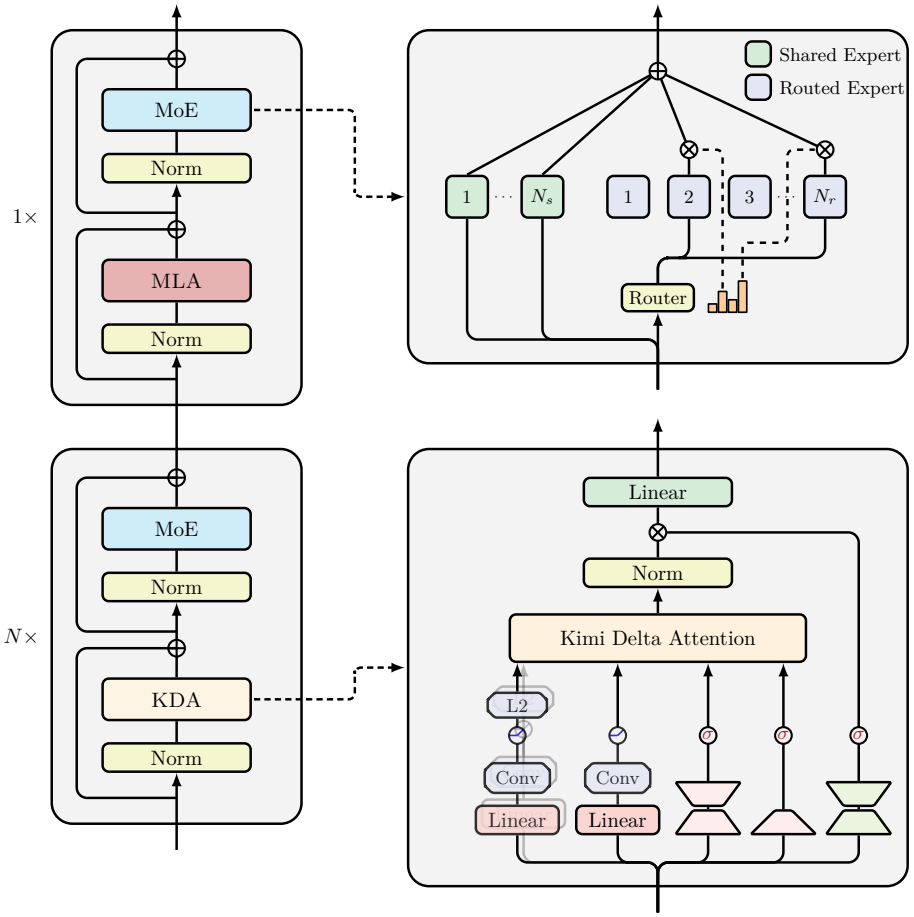


图 3: 我们 Kimi 线性模型架构的示意图，该架构由一系列包含 token 混合层和 MoE 通道混合层的块堆叠而成。具体而言，我们在每个块中交替插入 N KDA 层与一个 MLA 层进行 token 混合，其中在我们的实现中 N 设置为 3。

4 Kimi 线性模型架构

我们的模型架构的主要骨干遵循 Moonlight [**liu-2025-moonlight**]。
除了细粒度门控之外，我们还利用了多个组件来进一步提升 Kimi Linear 的表达能力。Kimi Linear 的整体架构如图 3 所示。

神经参数化 设 $\boldsymbol{x}_t \in \mathbb{R}^d$ 为第 t 个输入 token 的表示，KDA 中每个头 h 的输入计算如下

$$
\begin{aligned}
\boldsymbol{q}_t^h, \boldsymbol{k}_t^h &= \mathrm{L2Norm}(\mathrm{Swish}(\mathrm{ShortConv}(\mathbf{W}_{q/k}^h \boldsymbol{x}_t))) \in \mathbb{R}^{d_k} \\
\boldsymbol{v}_t^h &= \mathrm{Swish}(\mathrm{ShortConv}(\mathbf{W}_v^h \boldsymbol{x}_t)) \in \mathbb{R}^{d_v} \\
\boldsymbol{\alpha}_t^h &= f(\mathbf{W}_\alpha^\uparrow \mathbf{W}_\alpha^\downarrow \boldsymbol{x}_t) \in [0,1]^{d_k} \\
\beta_t^h &= \mathrm{Sigmoid}(\mathbf{W}_\beta^h \boldsymbol{x}_t) \in [0,1]
\end{aligned}
$$

其中 d_k, d_v 表示键和值头的维度，所有实验中均设置为 128。对于 $\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$，我们应用一个 ShortConv，随后是 Swish 激活，遵循 [**yang-2025-gdn**]。$\boldsymbol{q}$ 和 $\boldsymbol{k}$ 的表示进一步通过 L2Norm 进行规范化，以确保特征值的稳定性，如 [**yang-2024-parallelizing**] 所建议。每个通道的衰减 $\boldsymbol{\alpha}_t^h$ 通过低秩投影（$\mathbf{W}_\alpha^\downarrow$ 和 $\mathbf{W}_\alpha^\uparrow$，秩等于

头维度）以及与 GDN 和 Mamba [**yang-2025-gdn**, **mamba2**] 中使用的类似衰减函数 $f(\cdot)$ 参数化。在通过 $\mathbf{W}_o \in \mathbb{R}^{d\times d}$ 进行输出投影之前，我们使用通道级 RMSNorm [**zhang2019root**] 和一种数据相关的门控机制 [**qiu2025gated**]，其参数化形式为：

$$\boldsymbol{o}_t = \mathbf{W}_o \left(\text{Sigmoid}\left(\mathbf{W}_g^{\uparrow}\mathbf{W}_g^{\downarrow}\boldsymbol{x}_t\right) \odot \text{RMSNorm}\left(\text{KDA}\left(\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{\alpha}_t, \beta_t\right)\right)\right) \tag{10}$$

此处，输出门采用与遗忘门类似的低秩参数化，以确保参数量比较的公平性，同时保持与全秩门控相当的性能，并缓解注意力汇聚问题 [**qiu2025gated**]。非线性激活函数的选择将在 §5.2 中进一步讨论。

混合模型架构 长上下文检索仍然是纯线性注意力的主要瓶颈，因此我们将 KDA 与少量全量全局注意力（全 MLA）层进行混合 [**deepseekaiv3**]。对于 Kimi Linear，我们选择逐层方式（交替使用完整层）而非逐头方式（在层内混合注意力头），因为前者具有更优的基础设施简洁性和训练稳定性。实验表明，采用统一的 3:1 比例，即每 3 个 KDA 层对应 1 个全 MLA 层，能够实现最佳的质量-吞吐量权衡。我们在 § 7.2 中讨论了其他混合策略。

MLA 层不使用位置编码（NoPE）。 在 KIMI 线性中，我们对所有全注意力（MLA）层应用 NoPE。该设计将编码位置信息和近期性偏差（见 § 6.1）的全部责任交由 KDA 层承担。因此，KDA 被确立为主要的位置感知算子，其作用类似于或甚至强于诸如短卷积 [**allen2025physics**] 或 SWA [**puvvada2025swangpt**] 等辅助组件。我们的发现与先前结果 [**yang2025ropenopeagainnew**, **barbero2025round**, **deepseekaiv3**] 一致，后者同样表明，通过引入专门的位置感知机制来补充全局 NoPE 注意力，能够实现具有竞争力的长上下文性能。

我们注意到，NoPE 具有实际优势，尤其是在 MLA 中。首先，NoPE 可在推理过程中将其转换为高效纯多查询注意力（MQA）。其次，它简化了长上下文训练，因为无需调整 RoPE 参数，例如频率基底调优或 YaRN [**peng2023yarn**] 等方法。

5 实验

5.1 合成测试

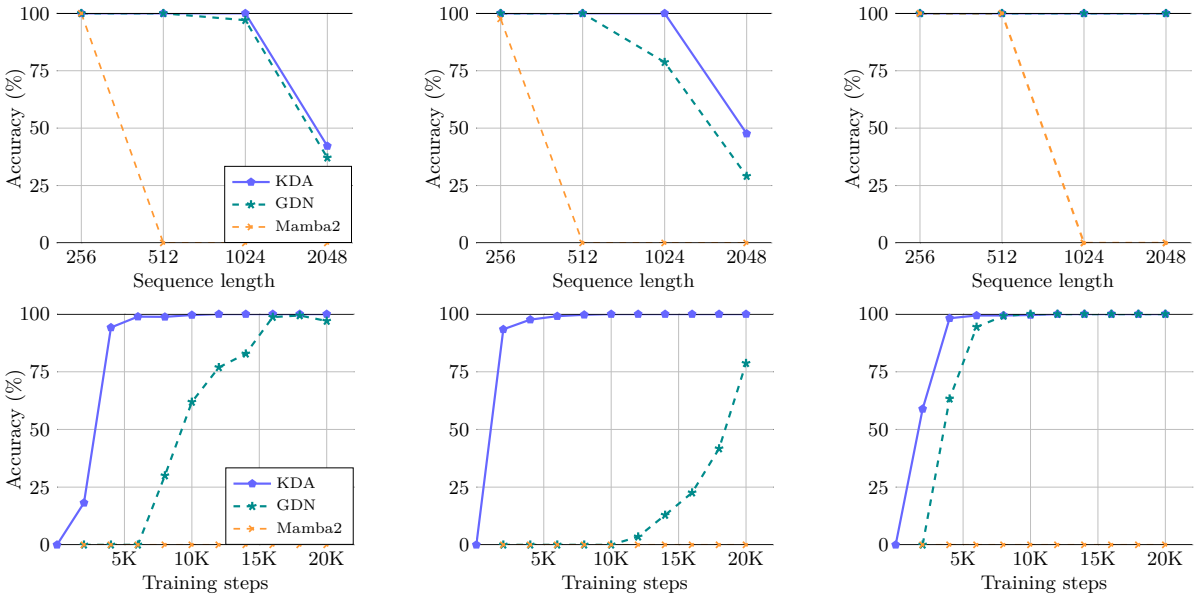
我们首先在三个合成任务上评估 KDA 与其他竞争性的线性注意力方法，作为长上下文性能的基准测试。在所有实验中，我们采用一致的模型配置：2 层，2 个注意力头，每个头的维度为 128。对于每个任务，我们最多训练 20,000 步，并对学习率进行 $\{5\times10^{-5}, 1\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}\}$ 范围内的网格搜索。随后，我们展示最佳训练准确率曲线。具体而言，我们比较两种场景：(1) 随着训练长度从 256 增加到 2,048 token，不同任务的性能表现，测量峰值准确率；(2) 在固定上下文长度为 1,024 token 的情况下，KDA、GDN 和 Mamba2 的收敛速度。

回文 回文任务要求模型将给定的随机 token 序列以逆序形式重现。如表 5.1 所示，对于输入 “O G R S U N E”，模型必须生成其确切的逆序。这类复制任务对线性注意力模型而言众所周知具有挑战性 [**jelassi-2024-repeat**]，因为它们难以从压缩的、固定大小的状态中精确地检索出完整的历史信息。

Input	O	G	R	S	U	N	E	<SEP>	E	N	U	S	R	G	O
Output	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	N	U	S	R	G	O	ϕ

多查询关联召回（MQAR） MQAR 评估模型在上下文中不同位置出现的多个查询时，检索相关值的能力。例如，如表 5.1 所示，模型需要为查询 B 回忆 0，为 G 回忆 5。该任务与语言模型化性能高度相关 [**arora-2023-zoology**]。

Input	A	1	C	3	B	0	M	8	G	5	E	4	<SEP>	B	G
Output	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	0	5



(a) 回文 (b) MQAR (c) 堆栈

图 4: 合成任务的结果：回文、多查询关联召回和状态追踪。

表 1: 对 KDA 与 MLA 注意力混合比例及其他关键组件的消融研究。我们列出了训练和验证困惑度（越低越好）以作比较。在最终实验中使用的最佳模型以灰色突出显示。

		Training PPL (↓)	Validation PPL (↓)
Hybrid ratio	3:1	**9.23**	**5.65**
	0:1	9.45	5.77
	1:1	9.29	5.66
	7:1	9.23	5.70
	15:1	9.34	5.82
w/o output gate		9.25	5.67
w/ swish output gate		9.43	5.81
w/o convolution layer		9.29	5.70

堆栈 我们通过模拟标准的后进先出（LIFO）栈操作来评估每个候选模型的状态追踪能力[**grazzi2025unlocking**]。我們的設定包含 64 個獨立的栈，每個栈由唯一的 ID 進行標識。模型需處理一連串兩種操作：1) PUSH：例如 "<PUSH> 1 G" 將元素 G 加入栈 1；2) POP：例如 "<POP> 0 E" 要求模型預測最近被推入栈 0 的元素 E。目標是准確追踪所有栈的狀態，在每次彈出請求時預測正確的元素。

图 4 显示了最终结果。在所有任务中，随着序列长度从 256 增加到 2,048 个 token，KDA 的准确率始终最高。特别是在回文和召回密集型 MQAR 任务中，KDA 的收敛速度显著快于 GDN。这证实了我们细粒度衰减的优势，它使模型能够有选择地遗忘无关信息，同时更精确地保留关键记忆。我们还观察到，在我们的模型情景下，典型的线性注意力模型 Mamba2 [**mamba2**]（仅使用乘法衰减且缺乏 delta 规则）在所有任务上均失败。

表 2: 用于缩放法则实验的模型配置和超参数。

# Act. Params.†	Head	Layer	Hidden	Tokens	lr	batch size‡
653M	16	16	1216	38.8B	2.006×10^{-3}	336
878M	18	18	1376	59.8B	1.790×10^{-3}	432
1.1B	20	20	1536	85.2B	1.617×10^{-3}	512
1.4B	22	22	1632	102.5B	1.486×10^{-3}	576
1.7B	24	24	1776	128.0B	1.371×10^{-3}	640

† Denotes the number of activated parameters in our MoE models, excluding embeddings.

‡ All models were trained with a context length of 4,096.

5.2 Kimi 线性模型关键组件消融实验

我们通过直接比较不同模型与第一尺度缩放法则模型（即 16 头、16 层）进行了一系列消融实验。所有模型均在相同的浮点运算预算和超参数下进行训练，以保证公平比较。我们在表 1中报告了训练和验证困惑度（PPL）。验证 PPL 是在一个高质量数据集上计算的，其分布与预训练语料库有显著差异，强调了在分布偏移下的泛化能力，因此也体现了训练和验证困惑度之间的差异。

输出门 我们将默认的 Sigmoid 输出门与两种变体进行比较：一种是没有门控的，另一种是采用 swish 门控的。结果表明，移除门控会降低性能。此外，[**yang-2025-gdn**] 采用的 swish 门控表现明显劣于 Sigmoid 。我们的观察与 [**qiu2025gated**] 的结论一致，他们同样认为 Sigmoid 门控具有更优的性能。因此，我们在所有实验中均采用 Sigmoid ，包括 GDN-H。

卷积层 轻量级的深度卷积使用较小的卷积核大小（例如，4）能够有效捕捉局部 token 依赖关系 [**allen2025physics**]，并被许多近期架构广泛采用 [**mamba2**, **arora-2023-zoology**, **yang-2024-parallelizing**]。我们在表 1 中验证了其有效性，表明卷积层在混合模型中仍发挥着不可忽略的作用。

杂交比例 我们进行了消融实验，以确定 KDA 线性注意力层与 MLA 全注意力层之间的最佳混合比例。在测试的多种配置中，3:1 的比例（每 1 层 MLA 层对应 3 层 KDA 层）取得了最佳效果，实现了最低的训练损失和验证损失。我们观察到其他比例存在明显的权衡：较高的比例（如 7:1）虽然训练损失相近，但验证性能显著下降；而较低的比例（如 1:1）虽保持相似的验证损失，却带来了更高的推理开销。此外，纯全注意力基准模型（0:1）表现较差。因此，3:1 的配置在模型性能与计算效率之间提供了最有效的平衡。

NoPE 与 RoPE 如表 5所示，Kimi Linear 在长上下文评估中始终表现优异，而 Kimi Linear (RoPE) 在短上下文任务上也取得了相似的得分。我们认为，这种发散现象源于位置偏置在深度方向上的分布方式不同。在 Kimi Linear (RoPE) 中，全局注意力层携带了强烈且显式的相对位置信号，而线性注意力（例如 GDN）则贡献了较弱的隐式位置归纳偏置。这种不匹配导致全局层对短距离顺序过度强调，虽然有利于短上下文，但在训练中期适应更长上下文时使模型灵活性下降。

相比之下，Kimi Linear 在各层间诱导了更为均衡的位置偏置，从而提升了长范围下的鲁棒性和外推能力，最终带来更强的长上下文性能。关于长上下文表现，如表 5所示，Kimi Linear 在不同长上下文基准上的平均得分最佳，验证了上一节中我们提出的优点。

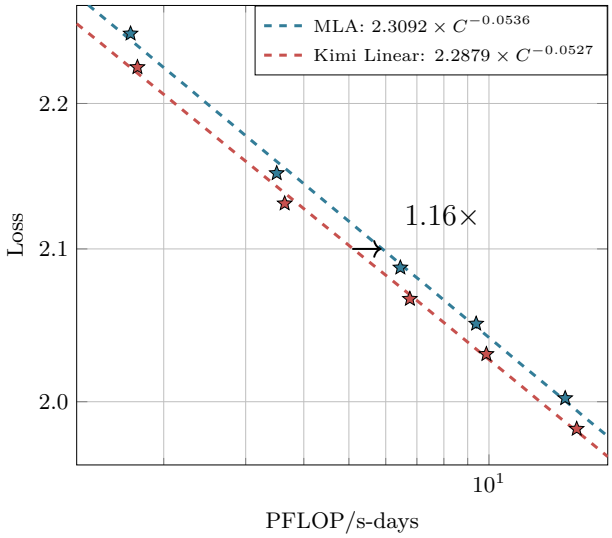


图 5: MLA 和 Kimi 线性拟合的缩放法则曲线。

5.3 Kimi 线性缩放法则

我们对一系列遵循 Moonlight [**liu-2025-moonlight**] 架构的 MoE 模型进行了缩放法则实验。在所有实验中，我们激活了 64 个专家中的 8 个，并使用了 Muon 优化器 [**liu-2025-moonlight**]。详细信息和超参数列于表 2。

对于 MLA，我们遵循 Chinchilla 缩放法则的方法论 [**hoffmann-2022-chinchilla**]，训练了五个不同大小的语言模型，并通过网格搜索仔细调优其超参数，以确保每个模型都能达到最佳性能。对于 KDA，我们保留了表 1中消融实验得出的最佳混合比例 3:1。除此之外，我们严格遵循 MLA 的训练配置，未作任何修改。如图 5所示，Kimi Linear 相较于计算最优训练的 MLA 基准模型，实现了 $\sim 1.16\times$ 的计算效率提升。我们预期，通过细致的超参数调优，KDA 将获得更优的缩放曲线。

5.4 实验设置

Kimi 线性与基准情景 我们针对一个全注意力的 MLA 基准和一个混合门控 DeltaNet（GDN-H）基准对我们的 Kimi 线性模型进行评估，所有模型均采用相同的架构、参数量和训练设置，以确保公平比较。该模型配置与 Moonlight [**liu-2025-moonlight**] 大致一致，主要区别在于 MoE 稀疏性提升至 32。每个模型在前向传播中激活 256 个专家中的 8 个，包括 1 个共享专家，总共包含 480 亿参数，每轮前向传播激活 30 亿参数。第一层实现为无 MoE 的稠密层，以确保训练稳定。为了评估 Kimi 线性模型中 NoPE 的有效性，我们还引入了一个使用 RoPE 的混合 KDA 基准，采用相同的模型配置，称为 Kimi 线性（RoPE）。

评估基准 我们的评估涵盖三大类基准，每一类均旨在评估模型的不同能力。

- **语言理解与推理**：Hellaswag [**zellers-2019-hellaswag**]，ARC-Challenge [**clark-2018-arc**]，Winogrande [**keisuke-2019-winogrande**]，MMLU [**hendrycks-2021-mmlu**]，TriviaQA [**joshi2017triviaqa**]，MMLU-Redux [**gema2024we**]，MMLU-Pro [**wang2024mmlu**]，GPQA-Diamond [**rein2024gpqa**]，BBH [**suzgun2022challenging**]，以及 [**white2024livebench**]。
- **代码生成**：LiveCodeBench v6 [4][**jain2024livecodebench**]，EvalPlus [**evalplus**]。
- **数学 & 逻辑**：AIME 2025，MATH 500，HMMT 2025，PolyMath-en。

[4]2024 年 8 月至 2025 年 5 月的问题

- **长上下文**: MRCR [5], RULER [**hsieh2024ruler**], 帧 [**krishna2024fact**], HELMET-ICL [**yen2025helmet**], RepoQA [**liu2024repoqa**], 长代码竞技场 [**bogomolov2024long**] 和 LongBench v2 [**bai2024longbench**]。
- **中文语言理解与推理**: C-Eval [**huang2023c**], 以及 CMMLU [**li-etal-2024-cmmlu**]。

评估配置 所有模型均使用温度 1.0 进行评估。对于方差较高的基准，我们报告 Avg@k 得分。对于基础模型，我们在 MMLU、MMLU-Redux、GPQA-Diamond 和 C-Eval 上采用基于困惑度的评估方法；其他情况下则采用生成式评估。为缓解 GPQA-Diamond 本身存在的高方差问题，我们报告八次独立运行的平均得分。所有评估均使用我们基于 LM-Harness-Evaluation [**biderman2024lessons**] 构建的内部框架进行，确保所有模型在一致的情景下进行评估。

5.4.1 预训练配方

预训练配方 所有模型均使用 4,096 token 的上下文窗口、MuonClip 优化器以及 WSD 学习率调度进行预训练，共处理来自 K2 预训练语料库的总计 1.4 万亿 token 样本 [**kimi2025k2**]。学习率设置为 1.1×10^{-3}，全局批量大小固定为 3200 万 token。它们还采用了与 Kimi K2 [**kimi2025k2**] 中确立的相同退火调度和长上下文活性值阶段。

我们最终发布的 Kimi Linear 检查点采用相同的预训练流程，但扩展了总计 5.7 万亿 token，以匹配 Moonlight 的预训练 token 数量。此外，最终检查点支持高达 100 万 token 的上下文长度。我们在附录 D中对比了 Kimi Linear@5.7T 与 Moonlight 的性能。

5.4.2 后训练配方

SFT 食谱 SFT 数据集在 Kimi K2 [**kimi2025k2**] SFT 数据的基础上，融入了额外的推理任务，构建了一个大规模的指令微调数据集，覆盖多个领域，尤其侧重于数学和编程。我们采用多阶段 SFT 方法，首先在广泛多样的 SFT 数据上训练模型，以提升其通用指令遵循能力，随后在计划性地针对推理密集型数据进行定向训练，以增强模型的推理能力。

强化学习配方 在强化学习训练提示集方面，我们主要整合了三个数据源：数学、代码和理工科。此项优化的主要目的是提升模型的推理能力。在进行强化学习之前，我们已预先筛选出与初始检查点难度适中相匹配的数据。

RL 训练的一个已知风险是通用能力可能出现退化。为缓解这一风险，我们在 RL 过程中引入 PTX 损失 [**ouyang2022training**]，遵循 K2 [**kimi2025k2**] 的做法。这包括在 RL 进程中对高质量、分布多样性的数据集同时进行 SFT。我们的 PTX 数据集涵盖了推理任务和通用任务。上述所有数据均为 K2 模型训练配方 [**kimi2025k2**] 的子集。

对于强化学习算法，我们采用与 K1.5 [**kimiteam2025kimik15scalingreinforcement**] 相同的算法，同时引入了几种高级技巧。我们注意到训练引擎与推理引擎之间的准确率不匹配可能导致强化学习学习不稳定。因此，我们引入了截断重要性采样，该方法能有效缓解回溯过程与训练之间的策略不匹配问题 [**yao2025offpolicy**]。此外，我们动态调整 KL 惩罚项和小批次大小（即每次迭代的更新次数），以确保强化学习训练的稳定性，并避免熵的坍缩 [**cui2025entropy**]。

[5] https://huggingface.co/datasets/openai/mrcr

5.5 主要结果

5.5.1 Kimi 线性的 @1.4T 结果

预训练结果 我们在表 3 中将我们的 Kini 线性模型与两个基准（MLA 和混合 GDN-H）进行了比较，使用了 1.4T 的预训练语料库。评估集中在三个领域：通用知识、推理（数学和代码）以及中文任务。在几乎所有类别中，Kimi 线性模型均持续优于两个基准。

- 通用知识：Kimi Linear 在 BBH、MMLU 和 HellaSwag 等关键基准测试中得分最高。
- 推理：它在数学（GSM8K）和大多数代码任务（CRUXEval）上表现领先，但在 EvalPlus 上的得分略低于 GDN-H。
- 中文任务：Kimi Linear 在 CEval 和 CMMLU 上取得最高得分。

总之，Kimi Linear 在短上下文预训练中表现出最强的性能，使其成为全注意力架构的一个有力替代方案。

表 3: Kimi Linear 与全注意力 MLA 基准以及混合 GDN 基准的性能对比，所有模型均采用相同的预训练方案。在短上下文预训练评估中，Kimi Linear 始终优于 MLA 和 GDN-H。每列最佳结果以 **粗体**标出。

	Type Base	MLA	GDN-H	Kimi Linear
	Trained Tokens	1.4T	1.4T	1.4T
General	HellaSwag	81.7	82.2	**82.9**
	ARC-challenge	64.6	66.5	**67.3**
	Winogrande	78.1	77.9	**78.6**
	BBH	71.6	70.6	**72.9**
	MMLU	71.6	72.2	**73.8**
	MMLU-Pro	47.2	47.9	**51.0**
	TriviaQA	68.9	70.1	**71.7**
Math & Code	GSM8K	83.7	81.7	**83.9**
	MATH	**54.7**	54.1	**54.7**
	EvalPlus	59.5	**63.1**	60.2
	CRUXEval-I-cot	51.6	56.0	**56.6**
	CRUXEval-O-cot	61.5	58.1	**62.0**
Chinese	CEval	79.3	79.1	**79.5**
	CMMLU	79.5	80.7	**80.8**

SFT 结果 Kimi Linear 在经过相同的监督微调（SFT）流程后，在通用任务和数学与代码任务上均表现出强劲性能，持续优于 MLA 和 GDN-H。在通用任务中，Kimi Linear 全面领先，在多个 MMLU 基准、BBH 和 GPQA-Diamond 上取得最高得分。在数学与代码任务中，它在 AIME 2025、HMMT 2025、PolyMath-en 和 LiveCodeBench 等高难度基准上均超越两个基准模型。尽管在 MATH500 和 EvalPlus 等少数任务上存在轻微例外，但 Kimi Linear 在各项任务中仍展现出稳健的优越性，证实其明显优于其他测试模型（GDN-H 和 MLA）。

长上下文性能评估 我们评估了 Kimi Linear 在 128k 上下文长度下的长上下文性能，对比了三种基准模型——MLA、GDN-H 和 Kimi Linear (RoPE)，涵盖多个基准测试（见表 5）。结果凸显了 Kimi Linear 在这些长上下文任务中的显著优势。其在所有任务中均持续优于 MLA 和 GDN-H，在 RULER（84.3）和 RepoQA（68.5）上取得了显著更高的得分。这一超越模式在大多数其他任务中均成立，仅在 LongBench V2 和 Frames

表 4: Kimi Linear 与全注意力 MLA 基准以及混合 GDN 基准的性能对比，所有模型均在预训练后使用相同的 SFT 配方。Kimi Linear 在短上下文指令微调基准上始终优于 MLA 和 GDN-H。每列最佳结果以 **粗体**标注。

	Type Instruct	MLA	GDN-H	Kimi Linear
	Trained Tokens	1.4T	1.4T	1.4T
General	BBH	68.2	68.5	**69.4**
	MMLU	75.7	75.6	**77.0**
	MMLU-Pro	65.7	64.8	**67.4**
	MMLU-Redux	79.2	78.7	**80.3**
	GPQA-Diamond (Avg@8)	57.1	58.6	**62.1**
	LiveBench (Pass@1)	45.7	**46.4**	45.2
Math & Code	AIME 2025 (Avg@64)	20.6	21.1	**21.3**
	MATH500 (Acc.)	80.8	**83.0**	81.2
	HMMT 2025 (Avg@32)	11.3	11.3	**12.5**
	PolyMath-en (Avg@4)	41.3	41.5	**43.6**
	LiveCodeBench v6 (Pass@1)	25.1	25.4	**26.0**
	EvalPlus	**62.6**	62.5	61.0

表 5: Kimi Linear 与 MLA、GDN-H 以及 Kimi Linear (RoPE) 在长上下文基准上的比较。最后一列报告了总体平均值 (↑)。所有模型均在 1.4T token 上进行训练。每列的最佳结果以 **粗体**标出。

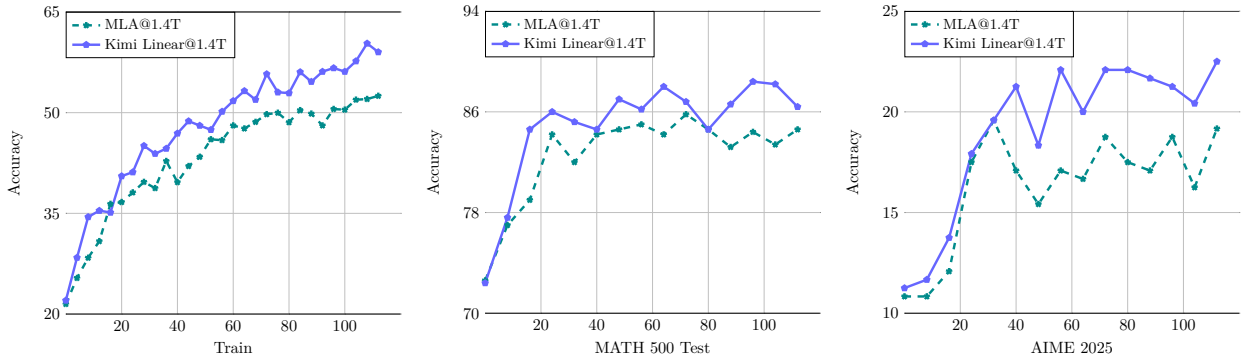
	RULER	MRCR	HELMET-ICL	LongBench V2	Frames	RepoQA	Long Code Arena		Avg.
							Lib	Commit	
MLA	81.3	22.6	88.0	**36.1**	**60.5**	63.0	32.8	**33.2**	52.2
GDN-H	80.5	23.9	85.5	32.6	58.7	63.0	34.7	30.5	51.2
Kimi Linear (RoPE)	78.8	22.0	88.0	35.4	59.9	66.5	31.3	32.5	51.8
Kimi Linear	**84.3**	**29.6**	**90.0**	35.0	58.8	**68.5**	**37.1**	32.7	**54.5**

上例外。总体而言，Kimi Linear 获得了最高的平均得分（54.5），进一步证明了其作为长上下文场景下领先注意力架构的有效性。

强化学习结果 为了比较 Kimi Linear 与 MLA 的强化学习收敛特性，我们使用内部数学训练集 [**kimi2025k2**] 进行 RLVR，并在数学测试集（如 AIME 2025、MATH500）上进行评估，同时保持算法和所有超参数相同，以确保性能比较的公平性。

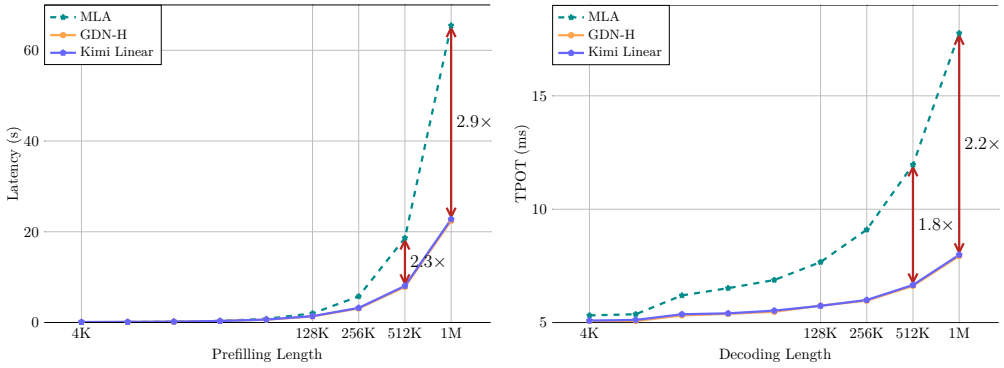
如图 6 所示，Kimi Linear 相较于 MLA 展现出更优的效率。在训练集上，尽管两个模型起始点相近，但 Kimi Linear 的训练准确率增长速率显著高于 MLA，且差距逐渐扩大。在测试集上也观察到类似现象。例如，在 MATH500 和 AIME2025 上，Kimi Linear 相较于 MLA 实现了更快且更优的提升。总体而言，在强化学习下的推理密集型长文本生成任务中，我们经验性地观察到，Kimi Linear 的表现显著优于 MLA。

总体发现总结在预训练和 SFT 阶段，性能表现呈现出清晰的层次：Kimi Linear 的表现优于 GDN-H，而 GDN-H 又优于 MLA。然而，在长上下文评估中，这一层次关系发生了变化。尽管 Kimi Linear 继续保持领先地位，但 GDN-H 的性能有所下降，排名落后于 MLA。此外，在强化学习阶段，Kimi Linear 同样展现出对 MLA 的优越性能。总体而言，Kimi Linear 在所有阶段均持续保持最佳表现，确立了其作为全注意力架构的更优替代方案的地位。



(a) (b) (c)

图 6: Kimi Linear@1.4T 和 MLA@1.4T 在数学强化学习训练过程中的训练准确率和测试准确率曲线。在整个强化学习过程中，Kimi Linear 始终显著优于完整的注意力基准。



(a) (b)

图 7: (a) MLA（全注意力）、混合 GDN-H 以及我们提出的 Kimi Linear 的预填充时间。(b) MLA、GDN-H 和 Kimi Linear 在解码过程中的每输出 token 时间（TPOT）。（此处测试使用批量大小 = 1。）

5.6 效率比较

预填充 & 解码速度 我们比较了全注意力 MLA [**deepseekaiv3**]、GDN-H 和 Kimi 线性模型在预填充和解码阶段的训练与解码时间，如图 7a 和图 7b 所示。需要注意的是，所有模型均基于 Kimi 线性 48B 情景，具有相同层数和注意力头数。

我们观察到：1) 尽管引入了更精细的衰减机制，Kimi 线性在预填充阶段相比 GDN-H 的延迟开销可忽略不计。如图 7a 所示，两者的性能曲线几乎无法区分，证实了我们的方法保持了高效率。随着序列长度增加，混合式 Kimi 线性模型相较于 MLA 基准展现出明显的效率优势。在较短序列长度（4k–16k）下，其性能与 MLA 相当，但从 128k 开始显著更快。在大规模场景下，该效率差距急剧扩大，对于 512k 序列，Kimi 线性比 MLA 快 2.3 倍；对于 1M 序列，快 2.9 倍。如图 1b 所示，Kimi 线性在解码阶段充分展现了其优势。在 1M 上下文长度的解码中，Kimi 线性比全注意力快 6× 倍。

表 6: 注意力机制的数学等价递归形式（$\boldsymbol{o}_t$）与并行形式（$\mathbf{O}$）的概述。为了获得更简洁的表示，我们省略了规范化项和 β_t。函数 ϕ 指对应于指数核的无穷维特征空间，即 $\phi(\boldsymbol{q})^\top \phi(\boldsymbol{k}) = \exp(\boldsymbol{q}^\top \boldsymbol{k})$。

	Recurrent form	Parallel form
SA [**vaswani-2017-attention**]	$\sum_{j=1}^{t} \exp\left(\boldsymbol{q}_t^\top \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\exp\left(\mathbf{QK}^\top\right) \odot \mathbf{M}\right) \mathbf{V}$
SA + RoPE [**su2024roformer**]	$\sum_{j=1}^{t} \exp\left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \boldsymbol{R}_s\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\exp\left(\mathbf{R}\left(\mathbf{Q}\right)\mathbf{R}\left(\mathbf{K}\right)^\top\right) \odot \mathbf{M}\right) \mathbf{V}$
LA [**wang-2020-linformer**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\mathbf{QK}^\top \odot \mathbf{M}\right) \mathbf{V}$
Mamba2 [**mamba2**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \alpha_s\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\mathbf{QK}^\top \odot \mathcal{A} \odot \mathbf{M}\right) \mathbf{V}$
GLA [**yang-etal-2024-gla**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \mathrm{Diag}\left(\boldsymbol{\alpha}_s\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\left(\mathbf{Q} \odot \right)\left(\underline{\mathbf{K}}\right)^\top \odot \mathbf{M}\right) \mathbf{V}$
DeltaNet [**schlag-2021-deltanet**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \left(\mathbf{I} - \boldsymbol{k}_s \boldsymbol{k}_s^\top\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\mathbf{QK}^\top \odot \mathbf{M}\right)\left(\mathbf{I} + \mathbf{KK}^\top \odot \mathbf{M}^-\right)^{-1} \mathbf{V}$
FoX [**lin2025forgetting**]	$\sum_{j=1}^{t} \exp\left(\boldsymbol{q}_t^\top \boldsymbol{k}_j\right) \left(\prod_{s=j+1}^{t} \alpha_s\right) \boldsymbol{v}_j$	$\left(\exp\left(\mathbf{QK}^\top\right) \odot \mathcal{A} \odot \mathbf{M}\right) \mathbf{V}$
DeltaFormer [**zhong2025understanding**]	$\sum_{j=1}^{t} \left(\phi(\boldsymbol{q}_t)^\top \left(\prod_{s=j+1}^{t} \left(\mathbf{I} - \phi\left(\boldsymbol{k}_s\right)\phi\left(\boldsymbol{w}_s\right)^\top\right)\right) \phi\left(\boldsymbol{k}_j\right)\right) \boldsymbol{v}_j$	$\left(\exp\left(\mathbf{QK}^\top\right) \odot \mathbf{M}\right)\left(\mathbf{I} + \exp\left(\mathbf{WK}^\top\right) \odot \mathbf{M}^-\right)^{-1} \mathbf{V}$
PaTH-FoX [**yang2025path**]	$\sum_{j=1}^{t} \exp\left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \left(\mathbf{I} - \boldsymbol{w}_s \boldsymbol{w}_s^\top\right)\right) \boldsymbol{k}_j\right) \left(\prod_{s=j+1}^{t} \alpha_s\right) \boldsymbol{v}_j$	$\left(\exp\left(\left(\mathbf{QK}^\top \odot \mathbf{M}\right)\left(\mathbf{I} + \mathbf{WW}^\top \odot \mathbf{M}^-\right)^{-1}\right) \odot \mathcal{A} \odot \mathbf{M}\right) \mathbf{V}$
GDN [**yang-2025-gdn**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \alpha_s \left(\mathbf{I} - \boldsymbol{k}_s \boldsymbol{k}_s^\top\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\mathbf{QK}^\top \odot \mathcal{A} \odot \mathbf{M}\right)\left(\mathbf{I} + \mathbf{KK}^\top \odot \mathcal{A} \odot \mathbf{M}^-\right)^{-1} \mathbf{V}$
Comba [**hu2025comba**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \left(\alpha_s - \boldsymbol{k}_s \boldsymbol{k}_s^\top\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\mathbf{QK}^\top \odot \mathcal{A} \odot \mathbf{M}\right)\left(\mathbf{I} + \mathbf{KK}^\top \odot \mathcal{A}^{i-1/j} \odot \mathbf{M}^-\right)^{-1} \mathbf{V}$
RWKV7 [**peng-2025-rwkv7**]	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \left(\mathrm{Diag}\left(\boldsymbol{\alpha}_s\right) - \left(\boldsymbol{b}_s \odot \hat{\boldsymbol{k}}_s\right) \hat{\boldsymbol{k}}_s^\top\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\left(\mathbf{Q} \odot \right)\left(\underline{\mathbf{K}}\right)^\top \odot \mathbf{M}\right)\left(\mathbf{I} + \left(\hat{\mathbf{K}} \odot {}^{0\to t-1}\right)\left(\frac{\hat{\mathbf{K}}}{\mathbf{B}}\right)^\top \odot \mathbf{M}^{-1}\right)^{-1} \mathbf{V}$
KDA (ours)	$\sum_{j=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{s=j+1}^{t} \mathrm{Diag}\left(\boldsymbol{\alpha}_s\right)\left(\mathbf{I} - \boldsymbol{k}_s \boldsymbol{k}_s^\top\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j$	$\left(\left(\mathbf{Q} \odot \right)\left(\underline{\mathbf{K}}\right)^\top \odot \mathbf{M}\right)\left(\mathbf{I} + \left(\mathbf{K} \odot \right)\left(\frac{\mathbf{K}}{\mathbf{B}}\right)^\top \odot \mathbf{M}^{-1}\right)^{-1} \mathbf{V}$

6 讨论

6.1 Kimi Delta Attention 作为可学习的位置嵌入

Transformer 中的标准注意力机制在设计上对输入序列的顺序是无感知的 [**vaswani-2017-attention**]，因此需要显式的位置编码 [**press-2022-alibi**, **shaw2018selfattention**]。在各种方法中，RoPE [**su2024roformer**] 因其有效性已成为现代大模型中的 *de facto* 标准 [**touvron-2023-llama**, **agarwal2025gpt**, **deepseekaiv3**]。像 RoPE 这样的乘法位置编码机制可以通过一种泛化注意力公式来分析：

$$s_{t,i} = \boldsymbol{q}_t^\top \left(\prod_{j=i+1}^{t} \mathbf{R}_j\right) \boldsymbol{k}_i \tag{11}$$

其中，第 t 个查询 $\boldsymbol{q}_t$ 与第 i 个键 $\boldsymbol{k}_i$ 之间的位置关系由累积的矩阵乘积反映。RoPE 将变换矩阵 $\mathbf{R}_j$ 定义为由 $d_k/2$ 个二维旋转矩阵 $\mathbf{R}_j^k = \begin{pmatrix} \cos(j\theta_k) & -\sin(j\theta_k) \\ \sin(j\theta_k) & \cos(j\theta_k) \end{pmatrix}$ 构成的块对角矩阵，其具有 **每二维**的角频率 θ_k 。由于旋转矩阵的性质，即 $\mathbf{R}_{t-i} = \mathbf{R}_t^\top \mathbf{R}_i$ ，绝对位置信息 $\mathbf{R}_t$ 和 $\mathbf{R}_i$ 可以分别应用于 $\boldsymbol{q}_t$ 和 $\boldsymbol{k}_i$ ，然后被转换为编码为 $\prod_{j=i+1}^{t} \mathbf{R}_j = \begin{pmatrix} \cos((t-i)\theta_k) & -\sin((t-i)\theta_k) \\ \sin((t-i)\theta_k) & \cos((t-i)\theta_k) \end{pmatrix}$ 的相对位置信息 $t-i$ 。

因此，我们表明，带有门控增量规则的线性注意力可以在公式 12 中以一种可比较的形式表达。其他注意力变体的类似形式总结在表 6 中。

$$\boldsymbol{o}_t = \sum_{i=1}^{t} \left(\boldsymbol{q}_t^\top \left(\prod_{j=i+1}^{t} \mathbf{A}_j \left(\mathbf{I} - \beta_j \boldsymbol{k}_j \boldsymbol{k}_j^\top\right)\right) \boldsymbol{k}_j\right) \boldsymbol{v}_j \tag{12}$$

从这一视角来看，GDN 可以被解释为一种乘法形式的位置编码，其状态转移矩阵依赖于数据，从而放松了 RoPE 所施加的正交性约束，可能具有更强的表达能力 [**yang2025path**]。[6] 这为解决 RoPE 已知的外推法问题提供了一种潜在方案，其固定的频率可能导致在训练过程中对上下文长度的过拟合 [**xiong-2023-llamalong**, **peng2023yarn**]。一些近期工作采用了诸如部分 RoPE [**barbero2025round**] 或甚至完全放弃显式位置编码（NoPE）[**kazemnejad2023impact**, **puvvada2025swangpt**, **deepseekaiv3**] 的变通方法。鉴于 GDN 在功能上与 RoPE 类似，我们在模型的全局全注意力层（MLA）中选择使用 NoPE，使位置信息能够由我们提出的 KDA 模型动态捕捉。

此外，RoPE 的一个关键优势在于其细粒度的位置编码，这是通过为每一对维度分配不同的旋转频率来实现的，这在特征维度上类似于非均匀傅立叶变换 [**barbero2025round**, **hua2024fourier**]。然而，标准的 GDN 采用的是每个头的标量衰减，缺乏这种通道间的多样性，这促使我们提出具有可学习通道门控的 KDA。

```
def chunk_dplr(q, k, v, a, b, g, chunk_size):
    B, H, T, K, V, BT = *q.shape, v.shape[-1], chunk_size
    NT, S = T // BT, k.new_zeros(B, H, K, V)
    q, k, v, a, b, g = map(lambda x: rearrange(x, 'b h (n c) d -> b h n c d', c=BT), [q, k, v, a, b, g])
    gc = g.cumsum(-2)
-   Aab, Aak, Aqb, Aqk = (torch.zeros(B, H, NT, BT, BT) for _ in range(4))

    for i in range(BT):
        a_i, q_i, g_i = (x[:,:,:,i,None] for x in (a, q, gc))
        mask = (torch.arange(BT) <= i)[..., None]
        s1_i = (g_i - gc).exp().where(mask, 0)
        s2_i= (g_i - g[:,:,:,i,None] - gc).where(mask, 0)
-       Aqk[..., i, :] = (q_i * k * s1_i).sum(-1)
-       Aqb[..., i, :] = (q_i * b * s1_i).sum(-1)
-       Aab[..., i, :] = (a_i * b * s2_i).sum(-1)
-       Aak[..., i, :] = (a_i * k * s2_i).sum(-1)
    for i in range(1, BT):
        Aab[..., i, :i] = Aab[..., i, :i] + (Aab[..., i, :, None] * Aab[..., :, :i]).sum(-2)
    Aab = Aab + torch.eye(BT)
    u, w = Aab @ (Aak @ v), Aab @ ((gc-g).exp() * a)
    o = torch.zeros_like(v)
    mask = torch.triu(torch.ones(BT, BT), diagonal=1)
    for i in range(0, NT):
        q_i, k_i, v_i, u_i, w_i, b_i = (x[:, :, i] for x in (q, k, v, u, w, b))
-       o1 = Aqk[:, :, i] @ v_i
-       o2 = Aqb[:, :, i] @ (u_i + w_i @ S)
-       o3 = (q_i * gkc[:, :, i].exp()) @ S
        o[:, :, i] = o1 + o2 + o3
        decay = (gc[:, :, i, -1, None] - gc[:, :, i]).exp()
        S = S * gc[:,:,i,-1,:,None].exp()
-       S +=(k_i * decay).transpose(-1,-2) @ v_i
-       S +=(b_i * decay).transpose(-1,-2) @ (u_i + w_i @ S)
    return o, S
```

(a) PyTorch 风格的分块 DPLR 伪代码。

```
def chunk_kda(q, k, v, a, b, g, chunk_size):
    B, H, T, K, V, BT = *q.shape, v.shape[-1], chunk_size
    NT, S = T // BT, k.new_zeros(B, H, K, V)
    q, k, v, g = map(lambda x: rearrange(x, 'b h (n c) ... -> b h n c ...', c=BT), [q, k, v, g])
    gc = g.cumsum(-2)
+   Aqk, Akk = (torch.zeros(B, H, NT, BT, BT) for _ in range(2))

    for i in range(BT):
        k_i, q_i = k[:, :, :, i, None], q[:, :, :, i, None]
        g_i = gc[...,i:i+1,:]
        mask = (torch.arange(BT) <= i)[..., None]
        s1_i = (g_i - gc).exp().where(mask, 0)
        s2_i = (gc - g_i).exp()
+       Aqk[:, :, :, i, :] = (q_i * k * s1_i).sum(-1)
+       Akk[..., i] = (k_i * k * s2_i).sum(-1)
    mask = torch.triu(torch.ones(BT, BT), diagonal=0)
    A = -Akk.masked_fill(mask, 0)
    for i in range(1, BT):
        A[..., i, :i] = A[..., i, :i] + (A[..., i, :, None] * A[..., :, :i].clone()).sum(-2)
    A = (A + torch.eye(BT))
    w, u = A @ (gc.exp() * k), A @ v
    o = torch.zeros_like(v)
    mask = torch.triu(torch.ones(BT, BT), diagonal=1)
    for i in range(0, NT):
        q_i, k_i, u_i, g_i, w_i = (x[:, :, i] for x in (q, k, u, gc, w))
+       o[:,:,i]=(q_i *g_i.exp()) @ S + Aqk @(u_i-w_i @ S)
        decay = (g_i[:,:,-1:] - g_i).exp()
        S = S * g_i[:, :, -1, :, None].exp()
+       S += (k_i * decay).transpose(-1,-2) @ v_i
    return o, S
```

(b) PyTorch 风格的分块 KDA 伪代码。

[6] 在保持正交性的情况下，绝对位置编码可以独立应用于 $\boldsymbol{q}$ 和 $\boldsymbol{k}$，这些编码在注意力计算 [**kexuefm-11033**] 过程中会自动转换为相对位置编码。

6.2 与 DPLR 的关系

(Gated) DeltaNet 可以推广到更具表现力的对角加低秩（DPLR）结构，定义为 $\mathbf{D} - \boldsymbol{a}_t \boldsymbol{b}_t^\top$。该结构也在 S4 [**gu-2022-efficiently**] 等模型中被探索，这些模型将静态 DPLR 形式用作状态转移矩阵。在计算过程中，该矩阵通常在复平面中进行联合对角化，从而将其表达能力限制为对角变换 [**merrill2024illusion**]。

尽管 DPLR 结构引入了更丰富的模型交互，并可通过其键值更新规则潜在提升召回率，但也存在明显局限性：计算成本高且难以并行化。这些缺点使得 DPLR 在大规模或实时场景中固有地较慢，参数效率的保持成为关键的设计挑战。

为解决此问题，KDA 引入了 DPLR 的约束变体，其中公式 1 可重写为 $\mathbf{S}_t = \left(\mathrm{Diag}(\boldsymbol{\alpha}_t) - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top \mathrm{Diag}(\boldsymbol{\alpha}_t)\right) \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$ 两者之间的对应关系为：

$$\mathbf{S}_t = (\mathbf{D} - \boldsymbol{a}_t \boldsymbol{b}_t^\top)\mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top, \text{s.t.}, \quad \mathbf{D} = \mathrm{Diag}(\boldsymbol{\alpha}_t), \boldsymbol{a}_t = \beta_t \boldsymbol{k}_t, \boldsymbol{b}_t = \boldsymbol{k}_t \odot \boldsymbol{\alpha}_t.$$

此外，通过共享 $\boldsymbol{\alpha}_t$，我们可以将其提取出来，如公式 1 所示，从而在 $\mathbf{S}_t$ 上实现细粒度的乘法衰减，方式类似于 GLA [**yang-etal-2024-gla**]，随后采用类似 DeltaNet [**schlag-2021-deltanet**, **yang-2024-parallelizing**] 的 Householder 变换以实现高效的状态更新。我们在清单 8a 和清单 8b 中提供了 DPLR 与 KDA 的分块式 PyTorch 风格伪代码实现的并排比较。主要改进如下：

- Listing 8a 第 13-16 行 与 Listing 8b 第 14-15 行：以分块形式表示的累积衰减项 $1/\Gamma$ 的倒数（公式 9）可能引入数值不稳定性。虽然我们可以通过二次分块 [**yang-2024-fla**] 来解决此问题，但这会带来额外的计算和 I/O 开销。通过在 DPLR 公式中固定 $\boldsymbol{a} = \boldsymbol{b} = \boldsymbol{k}$，KDA 消除了对两次二次分块步骤的需求，显著减少了冗余操作并提高了整体效率。
- 列表 8a 第 25-27 行，31-32 行 与列表 8b 第 26 行，29 行：KDA 在跨块计算和输出计算过程中进一步消除了大约三个矩阵乘法，从而实现了显著的内核级加速。

我们进一步在图 2 中对内核速度进行了基准测试，结果显示，在序列长度达到 64k 时，KDA 的速度几乎达到 DPLR 的 2×。

6.3 复杂性分析

训练失败 我们在 Kimi Linear 中保持与完整注意力 MLA 相似的参数数量。线性投影计算与全局注意力层保持一致。主要区别在于注意力计算相关的浮点运算次数（FLOPs）。为简化起见，我们仅关注非变长情况。基于门控规则核的实现，单个注意力头在 head_dim d_h 且固定块大小 $C = 64$ 时，门控增量规则 [**wang-deltanet**](per sequence of length T) 的理论 FLOPs 如下：

$$\mathrm{FLOPs}_{\mathrm{KDA}}(T; C, d_h) = 6Td_h^2 + 3TCd_h + TC^2. \tag{13}$$

对于完整的（全局）注意力，每个头的主导项是

$$\mathrm{FLOPs}_{\mathrm{Attn}}(T; d_h) = 2T^2 d_h. \tag{14}$$

推理策略与成本 Kimi Linear 中的推理策略采用混合方法，以优化计算和 I/O 效率。在预填充阶段，模型使用计算密集型的分块内核（见 § 3.1），而在自回归生成时切换到更高效的循环内核（式 2）。Linear KDA 的一个关键优势在于，其能够保持固定大小的状态（每头 $d_k \times d_v$，共 $d_k = d_v = 128$），而与序列长度无关。对于我们的混合模型，随着序列长度增加，I/O 限制的解码时间趋于达到相对于完整注意力的 3:1 最大混合效率比。这一趋势如图 7b 所示，Kimi Linear 在 1M token 上下文时实现了 2.3× 的加速。此外，通过消除对大型线性缩放 KV 缓存的需求，Kimi Linear 能够将内存资源重新分配给更大的批量大小，从而提升整体吞吐量。在长上下文场景（最高达 1M token）下，这种内存效率带来了理论上的解码速度提升，最高可达 6.3×（见图 1b）。

表 7: 在 TTT 框架下，通过状态更新规则及其学习目标的视角，概述不同注意力机制的对比 [**sun-2024-learning**]。为简洁起见，我们忽略所有归一化项和活性值/核函数。

	Objective $\mathcal{L}$	Update rule $\mathbf{S}_t = \mathbf{S}_{t-1} - \nabla_{\mathbf{S}_{t-1}}\mathcal{L}$
LA [**katharopoulos-2020-transformers**]	$-\langle \mathbf{S}_{t-1}^\top \boldsymbol{k}_t, \boldsymbol{v}_t \rangle$	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$
RetNet [**sun-2023-retnet**]	$-\beta_t \langle \mathbf{S}_{t-1}^\top \boldsymbol{k}_t, \boldsymbol{v}_t \rangle + \frac{1}{2} \left\| \sqrt{1-\alpha}\, \mathbf{S}_{t-1} \right\|_F^2$	$\mathbf{S}_t = \alpha \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$
Mamba2 [**mamba2**]	$-\beta_t \langle \mathbf{S}_{t-1}^\top \boldsymbol{k}_t, \boldsymbol{v}_t \rangle + \frac{1}{2} \left\| \sqrt{1-\alpha_t} \mathbf{S}_{t-1} \right\|_F^2$	$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$
GLA [**yang-etal-2024-gla**]	$-\langle \mathbf{S}_{t-1}^\top \boldsymbol{k}_t, \boldsymbol{v}_t \rangle + \frac{1}{2} \left\| \sqrt{\mathrm{Diag}\,(\mathbf{1}-\boldsymbol{\alpha}_t)} \mathbf{S}_{t-1} \right\|_F^2$	$\mathbf{S}_t = \mathrm{Diag}(\boldsymbol{\alpha}_t) \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$
HGRN2 [**qin-2024-hgrn2**]	$-\langle \mathbf{S}_{t-1}^\top (1-\boldsymbol{\alpha}_t), \boldsymbol{v}_t \rangle + \frac{1}{2} \left\| \sqrt{\mathrm{Diag}\,(\mathbf{1}-\boldsymbol{\alpha}_t)} \mathbf{S}_{t-1} \right\|_F^2$	$\mathbf{S}_t = \mathrm{Diag}(\boldsymbol{\alpha}_t) \mathbf{S}_{t-1} + (\mathbf{1}-\boldsymbol{\alpha}_t) \boldsymbol{v}_t^\top$
Longhorn [**liu-2024-longhorn**]	$\frac{1}{2} \left\| \boldsymbol{v}_t - \mathbf{S}_{t-1}^\top \boldsymbol{k}_t \right\|_{\mathrm{Diag}(\boldsymbol{\beta}_t)}^2$	$\mathbf{S}_t = \left(\mathbf{I} - \frac{\boldsymbol{\beta}_t}{1+\boldsymbol{\beta}_t \boldsymbol{k}_t^\intercal \boldsymbol{k}_t} \boldsymbol{k}_t \boldsymbol{k}_t^\intercal \right) \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\intercal$
Comba [**hu2025comba**]	$\frac{\beta_t}{2} \left\| \boldsymbol{v}_t - \mathbf{S}_{t-1}^\top \boldsymbol{k}_t \right\|^2 + \frac{1}{2} \left\| \sqrt{1-\alpha_t} \mathbf{S}_{t-1} \right\|_F^2$	$\mathbf{S}_t = \left(\alpha_t - \beta_t \boldsymbol{k}_t \hat{\boldsymbol{k}}_t^\top \right) \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$
RWKV7 [**peng-2025-rwkv7**]	$\frac{1}{2} \left\| \boldsymbol{v}_t - \mathbf{S}_{t-1}^\top \tilde{\boldsymbol{k}}_t \right\|^2 + \frac{1}{2} \left\| \sqrt{\mathrm{Diag}\,(\mathbf{1}-\boldsymbol{\alpha}_t)} \mathbf{S}_{t-1} \right\|_F^2$	$\mathbf{S}_t = \left(\mathrm{Diag}\,(\boldsymbol{\alpha}_t) - \left(\boldsymbol{b}_s \odot \hat{\boldsymbol{k}}_s \right) \hat{\boldsymbol{k}}_t^\top \right) \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$
GDN [**yang-2025-gdn**]	$\frac{\beta_t}{2} \left\| \boldsymbol{v}_t - \tilde{\mathbf{S}}_{t-1}^\top \boldsymbol{k}_t \right\|^2$	$\mathbf{S}_t = \left(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top \right) \alpha_t \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$
KDA (ours)	$\frac{\beta_t}{2} \left\| \boldsymbol{v}_t - \tilde{\mathbf{S}}_{t-1}^\top \boldsymbol{k}_t \right\|^2$	$\mathbf{S}_t = \left(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top \right) \mathrm{Diag}\,(\boldsymbol{\alpha}_t) \mathbf{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$

For GDN and KDA, the update can be viewed as performing an Stochastic Gradient Descent(SGD) process on the decayed state $\tilde{\mathbf{S}}$, that is, $\mathbf{S}_t = \tilde{\mathbf{S}}_{t-1} - \nabla_{\tilde{\mathbf{S}}_{t-1}} \mathcal{L}$, where $\tilde{\mathbf{S}}_{t-1}$ is decayed by scalar or fine-grained gate.

7 相关工作

7.1 高效的次二次注意力

基于 Transformer 的模型在处理长上下文时，标准自注意力机制 [**vaswani-2017-attention**] 的二次时间复杂度始终是一个根本性的瓶颈。随着大模型（LLMs）现在需要处理包含百万 token 的序列以完成智能体工具使用和仓库级代码分析等任务，这一限制变得愈发关键。[**deepseekaiv3**, **kimi2025k2**] 为克服这一挑战，大量研究探索了更高效的注意力机制 [**sun2025efficient**, **sun2025speedwinssurveyefficient**]，这些方法大致可分为两个主要方向：(1) 线性注意力，以及 (2) 稀疏注意力。

线性注意力 将二次注意力映射重构为核化特征交互，用正特征映射替代 Softmax，使得注意力可通过两次关联矩阵乘积 [**katharopoulos-2020-transformers**] 计算。这消除了显式的 $\mathcal{O}(T^2)$ 相似度矩阵，实现了与序列长度呈线性时间复杂度的计算。后续工作通过更精细的记忆控制显著增强了原始线性注意力，从与数据无关的“衰减” [**sun-2023-retnet**, **qin-2024-transnormerllm**] 转变为更具适应性的、依赖数据的机制 [**gu-2023-mamba**, **sun-2024-yoco**]，并将衰减粒度从粗粒度的头级 [**mamba2**] 提升至精确的通道级衰减。GLA 采用对角线、通道级门控机制，兼顾表达能力与效率，同时保持分块并行性 [**yang-2024-fla**, **yang-etal-2024-gla**]。表 7 总结了相应的更新规则。总体而言，这些方法将注意力建模为一种紧凑的递归记忆，通过并行前缀扫描算子和融合矩阵乘法进行更新，与现代加速器高度契合 [**hua-etal-2022-gau**]。

一种补充视角将线性注意力与快速权重记忆联系起来 [**schlag-2021-deltanet**]：状态是一个低容量的关联表，通过类似 Hebbian 的规则在线更新 [**munkhdalai-2019-metalearned**]，而慢权重则分摊何时存储、更新或遗忘的代价 [**munkhdalai2018metalearninghebbianfastweights**]。

在表 7 中，我们总结了现有的高效 token 混合方法，从状态更新机制和最优化目标两个角度进行了比较。

从这一视角来看，门控和衰减作为可学习的准则，能够缓解干扰并稳定最优化 [**sun-2024-learning**]。尽管取得了这些进展，线性注意力在极端长上下文检索中的确切复制和细粒度选择方面仍落后于完整注意力。这促使了混合设计（交错使用线性和完整注意力）以及更结构化的更新方式的发展。特别是，GDN/KDA 所采

用的门控增量规则为快速权重状态引入了秩-1 的修正更新，在保持操作符层面并行性的同时，提升了目标保留能力 [**yang-2024-parallelizing**]。

带门控机制的线性注意力 原始的线性注意力 [**katharopoulos-2020-transformers**] 众所周知缺乏 Softmax 注意力 [**vaswani-2017-attention**] 中固有的选择机制，表达能力不足。为解决此问题，门控线性注意力模型应运而生，成为内存高效且可并行化的替代方案 [**yang-2024-fla**, **yang-etal-2024-gla**, **gu-2023-mamba**]。这些模型不再存储不断扩展的键值缓存，而是采用固定大小的矩阵状态和可学习的门控机制，以选择性地保留或遗忘信息。该设计在保持推理时恒定的时间和内存复杂度的同时，实现了与 Softmax 注意力 [**merrill-sabharwal-2023-parallelism**, **zhong2025understanding**, **merrill2024illusion**] 相当的表达能力。此类模型用于记忆更新的通用递归形式 $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$ 可表示为：

$$\mathbf{S}_t = \mathbf{A}_t \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top, \quad \boldsymbol{o}_t = \mathbf{S}_t^\top \boldsymbol{q}_t. \tag{15}$$

各种门控线性注意力机制之间的主要区别在于遗忘门 $\mathbf{A}_t$ 的参数化方式，如表 7 所总结。例如，RetNet [**sun-2023-retnet**] 使用了与数据无关的标量衰减 α，而 Mamba2 [**mamba2**] 采用了依赖于数据的标量 α_t。具体而言，GLA [**yang-etal-2024-gla**] 采用了一个对角化的细粒度矩阵 $\mathrm{Diag}(\boldsymbol{\alpha}_t) \in \mathbb{R}^{d_k \times d_k}$，能够在效率与性能之间提供有效的权衡。其他变体如表 7 所示。

稀疏注意力 另一项独立的研究工作通过利用注意力机制固有的稀疏性，降低了标准注意力的二次复杂度，通过在经过策略性选择的 token 子集上执行计算来近似完整的注意力得分。核心挑战在于如何有效识别这一子集，同时不损害模型性能。早期方法通常采用高效且无需训练的静态模式，例如滑动窗口和扩张窗口 [**ding2023longnetscalingtransformers1000000000**, **gu2025attentionsinkemergeslanguage**, **xiao2023efficient**]，或固定模式 [**zaheer2020big**, **guo2019star**]，但其僵化的结构常常影响模型准确率。更先进的方法则基于上下文动态确定重要位置，如聚类 [**kitaev2020reformer**, **wu2022memorizing**] 和轻量级路由机制 [**fu2024moa**, **pikekos2025mixture**, **ainslie2023colt5**, **bertsch2023unlimiformer**]，但这种动态选择过程引入了计算开销，可能导致它们在没有专用内核加速的情况下无法实现其理论上的提速 [**dong2024flexattentionprogrammingmodel**]。部分模型还在推理阶段进一步引入无需训练的稀疏化 [**xiao2023efficient**, **xu2025xattention**]。

最近的稀疏注意力方法开始优先考虑硬件协同设计，如 NSA [**yuan2025nativesparseattentionhardwarealigned**, **minicpmteam2025minicpm4ultraefficientllmsend**] 和 MoBA [**lu2025mobamixtureblockattention**]，二者均从 token 级选择转向块级选择。在 NSA 中，每个查询根据由 MLP 生成的得分动态选择块。该方法的效率依赖于其采用的分组查询注意力（GQA）[**touvron-2023-llama**]，并使用较大的头数量（通常为 16 的倍数），这一配置专门设计用于通过高度并行化的张量–矩阵乘法加速计算。类似地，MoBA 执行 top-k 块选择，但通过 flash-attention 内核 [**dao-2022-flashattn**] 高效计算 log-sum-exp（LSE）得分。与 NSA 和 MoBA 不同，近期提出的 DeepSeek-V3.2-Exp 注意力（DSA）[**deepseekai2024deepseekv32**] 重新引入了 token 级稀疏性，通过使用低精度 fp8 实现的可学习全注意力索引器以及较小的头维度来完成 token 选择，从而保持高效。

讨论 线性注意力和稀疏注意力代表了高效长上下文建模的两种不同路径。稀疏注意力在更有效地检索细粒度历史信息方面表现更优，但这一优势伴随着需要存储完整的键值（KV）缓存以进行 token 选择，导致其效率低于维持恒定状态的线性注意力模型。此外，稀疏注意力仅执行信息选择，其理论表达能力上限仍受限于全注意力模型。相比之下，线性注意力基于“压缩即智能”的原则，能够以固定大小的状态实现泛化，并且在结合 Delta 学习规则时，理论上可具备更强的表达能力。尽管线性注意力传统上因检索能力较弱而受到批评，但这一局限可通过状态扩展 [**du2025mom**, **guo2025log**, **yau2025sequential**, **hu2024attractor**] 或相关技术加以缓解。然而，尽管具有这些优势，线性注意力仍受限于当前硬件实现以及缺乏优化的推理基础设施。我们的工作通过 Kimi Linear 模型克服了这些限制，该模型集成了 vLLM 以实现高效推理。我们提出的 KDA 在与全注意力基准相比时表现出具有竞争力的性能（表 3），并在一百万 token 上下文场景下实现

了超过 2× 的解码加速（图 7b）。尽管线性注意力与稀疏注意力在高效长上下文建模方面采取了不同的方法，二者并非互斥。未来的工作可以探索融合两者优势的混合模型，利用线性注意力的压缩与泛化能力，结合稀疏注意力在细粒度信息检索方面的优势，进一步提升模型性能与效率。

7.2 混合模型

尽管效率较高，纯线性注意力在精确的记忆检索和确切复制方面仍然存在困难 [**jelassi-2024-repeat**, **wen2024rnns**] 这一缺陷阻碍了其在工业级大模型中的应用，因为在这些场景中，强大的长上下文召回能力（例如超过 100 万 token）以及在大型代码仓库上可靠地使用工具至关重要 [**kimi2025k2**] 最近的研究表明，线性注意力与全注意力能够有效互补，从而催生了多种混合设计。

层内杂化 一种混合架构的类别是层内混合架构，它在每一层内自适应地融合不同机制的输出。一种常见的实现方式是在每一层内融合异构头的输出，例如将标准注意力与状态空间模型（SSMs）结合 [**dong2024hymba**, **li2025transmamba**]。相比之下，序列级方法则对输入的不同部分应用不同的机制。例如，某些方法对历史上下文使用线性注意力，对近期 token 使用 SWA [**zhang2024lolcats**, **lan2025liger**, **munkhdalai2024leave**]；而 NHA [**du2025native**] 使用 GSA [**zhang2024gated**] 压缩历史信息，并将其与局部滑动窗口上下文结合，以模拟标准注意力操作。

层间混合 混合层内部的一个主要缺点是系统复杂度增加和推理开销上升。异构机制需要分离的计算路径，这使得诸如分布式并行等优化变得更加复杂。为缓解这一挑战，层间混合已成为大模型中更为广泛采用且实用的策略 [**minimax2025minimax01**, **lieber2024jamba**, **team2025hunyuan**]。该方法涉及以预定义的比例堆叠不同的层类型，例如完整的注意力层和一种线性替代方案。基于这一范式，我们实现了一种简单而有效的策略：以固定的 3:1 比例交错排列线性和完整注意力层（详见 § 5.2 的消融实验）。这种规则且重复的结构简化了键值缓存管理，并能与标准优化无缝集成。对于混合结构中的线性部分，我们摒弃了使用 Mamba2 [**mamba2**] 的常见做法。相反，我们采用了 KDA，因为我们发现它在整体性能上表现更优，尤其是在检索和复制能力方面。

讨论 最近的研究表明，混合模型对 RoPE 基频的调整较为敏感，这一脆弱性使得上下文窗口扩展变得复杂 [**zuo2025falconh1familyhybridheadlanguage**]。这种敏感性可能阻碍模型对更长序列的外推能力。为应对这一挑战，近期模型趋向于采用不包含位置嵌入（NoPE）的解决方案。例如，Falcon-H [**zuo2025falconh1familyhybridheadlanguage**] 使用了一个非传统的高基频（如 $b \approx 10^{11}$ ），使其位置编码接近于无位置嵌入状态。在架构上，SwanGPT [**puvvada2025swangpt**] 将基于 RoPE 的层与基于 NoPE 的全注意力层交错排列。遵循这一方向，我们发现将 KDA 层与 NoPE 全注意力相结合也是一种非常有效的策略，能够实现简单的上下文窗口扩展。

结论

我们引入了 Kimi Linear，这是一种混合线性注意力架构，旨在满足智能体智能和测试时缩放的效率需求，同时不牺牲质量。Kimi Linear 的核心是 Kimi Delta Attention (KDA)，这是一种先进的线性注意力模块，具有通道级门控机制，可增强记忆控制，并在混合架构中实现类似 RNN 的模型。通过以 3:1 的比例交错使用 KDA 和全局注意力，Kimi Linear 将内存使用量最多减少 75%，同时实现高达 6.3× 的解码吞吐量提升，并优于全注意力基准。我们的方法为大规模语言模型提供了一种可扩展、高效的解决方案，开源的 KDA 内核和预训练检查点有助于进一步研究。

A 贡献

作者按贡献重要性排序，项目领导角色的人员列在最后。该项目由 Moonshot AI 开发，其中包含几位外部合作者，用 # 标注。带星号 (*) 标记的姓名表示该人员已不再属于我们的团队。

张宇 [1]
林宗宇 *
姚兴成
胡家熙 [2]
孟凡清
刘承荫
新门
杨松林 # [3]
李志远
李文涛
陆恩哲
刘伟州
陈燕如
徐伟信
余龙辉
王业杰
余凡
钟光政
袁恩明
张德浩
张一志
刘天佑
王海明
方升君
何伟然
刘少伟
李一伟
苏建林
邱杰中 [4]
庞博
严俊杰
姜哲君
黄伟晓
Bohong Yin
尤佳诚
楚伟
王正涛
赵宏
陈玉田
陈冠多
王玉成
郑华斌
王峰
刘一博
董梦南
郑张
潘思源
吴文浩
吴宇浩
管龙宇
陶佳文
傅国宏 # [1]
许欣然
王玉芝
赖国坤
吴裕新
周信宇
杨志林
杜玉伦

[1] 苏州大学，中国

[2] 香港科技大学（广州）

[3] 麻省理工学院

[4] 中国科学院杭州医学研究所

B KDA 分块并行的推导

我们首先回顾 KDA 的递归形式：

$$\mathbf{S}_{[t]}^r = \underbrace{\left(\prod_{i=1}^{t}\left(\mathbf{I}-\beta_{[t]}^i \boldsymbol{k}_{[t]}^i \boldsymbol{k}_{[t]}^{i\top}\right)\mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^i)\right)}_{:=\mathbf{P}_{[t]}^r}\cdot\mathbf{S}_{[t]}^0 + \underbrace{\sum_{i=1}^{r}\left(\prod_{j=i}^{t}\left(\mathbf{I}-\beta_{[t]}^j \boldsymbol{k}_{[t]}^j \boldsymbol{k}_{[t]}^{j\top}\right)\mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^j)\right)\cdot\beta_{[t]}^i \boldsymbol{k}_{[t]}^i \boldsymbol{v}_{[t]}^{i\top}}_{:=\mathbf{H}_{[t]}^r}$$

$$= \mathbf{P}_{[t]}^r\cdot\mathbf{S}_{[t]}^r + \mathbf{H}_{[t]}^r$$

我们的目标是将 $\mathbf{P}_{[t]}^r$ 和 $\mathbf{H}_{[t]}^r$ 变换为适合并行计算的矩阵形式。

我们表明，$\mathbf{P}_{[t]}^r$（涉及广义 Householder 矩阵的累积乘积）可以使用经典的 WY 表示进行优化。

Proposition 1. 矩阵 $\mathbf{P}_{[t]}^r$ 可以表示为：

$$\mathbf{P}_{[t]}^r = \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r}\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^{i\to r})\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} \tag{16}$$

其中辅助向量 $\boldsymbol{w}_{[t]}^r \in \mathbb{R}^{d_k}$ 通过以下循环关系计算得到：

$$\boldsymbol{w}_{[t]}^r = \beta_{[t]}^r\left(\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r)\boldsymbol{k}_{[t]}^r - \sum_{i=1}^{r-1}\boldsymbol{w}_{[t]}^i\left(\boldsymbol{k}_{[t]}^{i\top}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^r\right)\right) \tag{17}$$

证明. 我们采用数学归纳法进行证明。

归纳步骤：假设该命题对 $r-1$ 成立，即 $\mathbf{P}_{[t]}^{r-1} = \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^{r-1}) - \sum_{i=1}^{r-1}\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^{i\to r-1})\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top}$ 。我们现在推导出：

$$\begin{aligned}
\mathbf{P}_{[t]}^r &= \left(\mathbf{I}-\beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\right)\mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^r)\mathbf{P}_{[t]}^{r-1} \\
&= \left(\mathbf{I}-\beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\right)\mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^r)\left(\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^{r-1}) - \sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r-1}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top}\right) \\
&= \left(\mathbf{I}-\beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\right)\left(\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top}\right) \\
&= \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} - \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) + \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} \\
&= \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} - \boldsymbol{k}_{[t]}^r\left(\beta_{[t]}^r\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r)\boldsymbol{k}_{[t]}^r\right)^\top + \boldsymbol{k}_{[t]}^r\left(\beta_{[t]}^r\sum_{i=1}^{r-1}\boldsymbol{w}_{[t]}^i\left(\boldsymbol{k}_{[t]}^{i\top}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^r\right)\right)^\top \\
&= \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} - \boldsymbol{k}_{[t]}^r\underbrace{\left(\beta_{[t]}^r\left(\mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r)\boldsymbol{k}_{[t]}^r - \sum_{i=1}^{r-1}\boldsymbol{w}_{[t]}^i\left(\boldsymbol{k}_{[t]}^{i\top}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^r\right)\right)\right)^\top}_{\boldsymbol{w}_{[t]}^r} \\
&= \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r-1}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top} - \boldsymbol{k}_{[t]}^r \boldsymbol{w}_{[t]}^{r\top} \\
&= \mathrm{Diag}(\boldsymbol{\gamma}_{[t]}^r) - \sum_{i=1}^{r}\mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right)\boldsymbol{k}_{[t]}^i \boldsymbol{w}_{[t]}^{i\top}
\end{aligned}$$

归纳步骤成立。 ■

类似于 $\mathbf{P}_{[t]}^r$ ， $\mathbf{H}_{[t]}^r$ 也可以表示为可并行的形式。

Proposition 2. 矩阵 $\mathbf{H}_{[t]}^r$ 可以表示为：

$$\mathbf{H}_{[t]}^r = \sum_{i=1}^{r} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} \tag{18}$$

其中辅助向量 $\boldsymbol{u}_{[t]}^r \in \mathbb{R}^{d_v}$ 通过以下循环关系计算得到：

$$\boldsymbol{u}_{[t]}^r = \beta_{[t]}^r \left(\boldsymbol{v}_{[t]}^r - \sum_{i=1}^{r-1} \boldsymbol{u}_{[t]}^i \left(\boldsymbol{k}_{[t]}^{i\top} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^r \right) \right) \tag{19}$$

证明. 我们再次使用数学归纳法。

归纳步骤： 假设该命题对 $r-1$ 成立。

$$\begin{aligned}
\mathbf{H}_{[t]}^r &= \left(\mathbf{I} - \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\right) \mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^r) \mathbf{H}_{[t]}^{r-1} + \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{v}_{[t]}^{r\top} \\
&= \left(\mathbf{I} - \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\right) \mathrm{Diag}(\boldsymbol{\alpha}_{[t]}^r) \left(\sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r-1}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} \right) + \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{v}_{[t]}^{r\top} \\
&= \left(\mathbf{I} - \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top}\right) \left(\sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} \right) + \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{v}_{[t]}^{r\top} \\
&= \sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} - \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{k}_{[t]}^{r\top} \sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} + \beta_{[t]}^r \boldsymbol{k}_{[t]}^r \boldsymbol{v}_{[t]}^{r\top} \\
&= \sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} - \boldsymbol{k}_{[t]}^r \left(\beta_{[t]}^r \sum_{i=1}^{r-1} \left(\boldsymbol{k}_{[t]}^{r\top} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \right) \boldsymbol{u}_{[t]}^i \right)^\top + \boldsymbol{k}_{[t]}^r \beta_{[t]}^r \boldsymbol{v}_{[t]}^{r\top} \\
&= \sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} + \boldsymbol{k}_{[t]}^r \left(\underbrace{\beta_{[t]}^r \left(\boldsymbol{v}_{[t]}^r - \sum_{i=1}^{r-1} \boldsymbol{u}_{[t]}^i \left(\boldsymbol{k}_{[t]}^{i\top} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^r \right) \right)}_{\boldsymbol{u}_{[t]}^r} \right)^\top \\
&= \sum_{i=1}^{r-1} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top} + \boldsymbol{k}_{[t]}^r \boldsymbol{u}_{[t]}^{r\top} \\
&= \sum_{i=1}^{r} \mathrm{Diag}\left(\boldsymbol{\gamma}_{[t]}^{i\to r}\right) \boldsymbol{k}_{[t]}^i \boldsymbol{u}_{[t]}^{i\top}
\end{aligned}$$

归纳步骤成立。 ■

C KDA 分块伪代码

```python
def chunk_kda(
    q: torch.Tensor,
    k: torch.Tensor,
    v: torch.Tensor,
    g: torch.Tensor,
    beta: torch.Tensor,
    initial_state: Optional[torch.Tensor] = None,
    chunk_size: int = 64
):
    dtype = v.dtype
    B, T, H, K, V, C = *q.shape, v.shape[-1], chunk_size
    N = T // C

    q, k, v, g, beta = map(
        lambda x: rearrange(x, 'b (n c) h ... -> b h n c ...', c=C).to(torch.float),
        [q, k, v, g, beta]
    )
    q = q * K**-0.5
    g = g.cumsum(-2)
    mask = torch.triu(torch.ones(C, C, dtype=torch.bool, device=q.device), diagonal=0)

    A = torch.zeros(B, H, N, C, C, dtype=torch.float, device=q.device)
    for i in range(C):
        k_i = k[..., i, :]
        g_i = g[..., i:i+1, :]
        A[..., i] = torch.einsum('... c d, ... d -> ... c', k * (g - g_i).exp(), k_i)
    A = A * beta[..., None]
    # matrix inverse by forward substitution
    A = -A.masked_fill(mask, 0)
    for i in range(1, C):
        A[..., i, :i] = A[..., i, :i].clone() + (A[..., i, :, None].clone() * A[..., :, :i].clone()).sum(-2)
    A = (A + torch.eye(C, dtype=torch.float, device=q.device)) * beta[..., None, :]

    w = A @ (g.exp() * k)
    u = A @ v

    S = k.new_zeros(B, H, K, V)
    if initial_state is not None:
        S += initial_state
    o = torch.zeros_like(v)
    # strictly lower triangular
    mask = torch.triu(torch.ones(C, C, dtype=torch.bool, device=q.device), diagonal=1)
    for i in range(0, N):
        # [B, H, C, ...]
        q_i, k_i, u_i, g_i, w_i = q[:, :, i], k[:, :, i], u[:, :, i], g[:, :, i], w[:, :, i]
        A = torch.zeros(B, H, C, C, dtype=torch.float, device=q.device)
        # secondary chunking for numerical stability
```

```
        for j in range(C):
            k_j = k[:, :, i, j]
            g_j = g[:, :, i, j:j+1, :]
            A[..., j] = torch.einsum('... c d, ... d -> ... c', q_i * (g_i - g_j).exp(), k_j)
        A = A.masked_fill(mask, 0)
        v_i = u_i - w_i @ S
        o[:, :, i] = (q_i * g_i.exp()) @ S + A @ v_i
        S = S * rearrange(g_i[:, :, -1].exp(), 'b h k -> b h k 1')
        S += rearrange((g_i[:, :, -1:] - g_i).exp() * k_i, 'b h c k -> b h k c') @ v_i
    return rearrange(o, 'b h n c d -> b (n c) h d').to(dtype)

```

Listing 1: KDA 分块形式的伪 PyTorch 风格代码片段。

D Kimi 线性的 @5.7T 结果

在 Moonlight 的基础上，我们还使用扩展的 5.7 万亿 token 数据集训练了 Kimi Linear，以展示其有效性。在 3× 稀疏性以及新的注意力架构设计下，Kimi Linear 在几乎所有基准测试中均持续优于 Moonlight，凸显了新架构的有效性。基础模型的结果如表 8所示，指令微调模型的结果如表 9所示。由于超出 8K 上下文长度限制，Moonlight-Instruct 在相关任务上未进行评估（“-”）。

Kimi Linear@5.7T 在 1M 上下文长度的 RULER 测试中获得了 94.8 的得分。这一长上下文表现进一步证明，Kimi Linear 是全注意力架构的一个有前景的替代方案，在提供相当或更优结果的同时，可能实现更高效的资源利用。

表 8: Kimi-Linear-Base 与 Moonlight-Base 在各类任务中的表现。

	Benchmark	#Shots	Kimi-Linear-Base	Moonlight-Base
	Architecture	-	MoE	MoE
	# Activated Params	-	3B	3B
	# Total Params	-	48B	16B
	Trained Tokens	-	5.7T	5.7T
General	TriviaQA	5-shots	75.2	66.2
	SimpleQA	5-shots	10.1	5.6
	MMLU-Pro	5-shots	54.8	42.4
	MMLU-redux	5-shots	79.7	73.8
	WinoGrande	5-shots	81.5	74.6
	GPQA-Diamond (avg@8)	5-shots	40.4	35.2
Math	MATH	4-shots	58.5	45.3
	GSM8k	8-shots	86.3	77.2
	GSM8k-platinum	8-shots	89.6	79.4
	CMATH	6-shots	85.5	79.6
Code	CRUXEval-I-cot	0-shots	61.0	45.9
	CRUXEval-O-cot	0-shots	67.0	46.6
	LiveCodeBench (v6)	1-shots	20.0	14.3
	EvalPlus	-	64.9	50.3
Chinese	C-Eval	5-shots	83.3	77.6
	CSimpleQA	5-shots	53.5	34.7

表 9: Kimi-Linear-Instruct 与 Moonlight-Instruct 在各类任务中的表现。

	Benchmark	Kimi-Linear-Instruct	Moonlight-Instruct
	Architecture	MoE	MoE
	# Activated Params	3B	3B
	# Total Params	48B	16B
	Trained Tokens	5.7T	5.7T
General	RULER@128k	95.4	-
	RULER@1M	94.8	-
	GPQA-Diamond (Avg@8)	71.7	24.7
	MMLU-Redux (EM)	86.9	66.9
	MMLU-Pro (EM)	72.7	43.8
	FaithJudge (1-Hallu.)	64.2	56.0
Math	AIME 2025 (Avg@64)	58.6	-
	MATH500 (Acc.)	94.6	58.0
	HMMT 2025 (Avg@32)	44.5	-
Code	LiveCodeBench v6 (Pass@1)	45.7	11.9
	OJBench (Pass@1)	14.2	-
	Humaneval$^+$	70.9	46.3
	MBPP$^+$	72.4	56.3