One Line of Code to Translate arXiv Papers into Chinese

Jianlin Su

2026-01-28

Introduction

Long-time readers might know that the author considers himself a relatively firm "traditional programming enthusiast" — still not using IDEs, code completion, and only requires syntax highlighting in the editor. Even the articles on the website "Scientific Space" are written by typing the HTML source code manually (though this is partly due to historical reasons). The entire codebase of Cool Papers, from frontend to backend, was also written by hand.

However, even someone as "stubborn" as the author must admit: for certain tasks, completing them through a Code Agent offers unparalleled advantages — even to the point where you'd feel at a loss trying to do it manually. This article presents a classic example: translating an arXiv paper with just one line of code.

1 The Method

First, deploy kimi-cli locally, configure it to use kimi-k2.5, and install a LaTeX compilation environment (the author uses MacTeX). Then, download the paper source from arXiv, extract it, navigate into the source directory, and execute:

```
kimi -print -prompt "You are currently in the directory of a LaTeX
source for an English paper.  Your task is to translate the paper into
Chinese.  All English text should be translated into Chinese, while
formulas remain unchanged.  Translate only necessary captions for tables
and figures, and only translate necessary comments in code blocks or
algorithm boxes -- do not translate names.  Save the translated source
into a new directory named 'paper_cn' in the current directory, ensuring
the source remains compilable.  After translation, carefully check
for any untranslated sections or files.  Finally, recompile the translated
result into a PDF using xelatex.  If you encounter Chinese encoding
issues during compilation, consider using usepackage[fontset=fandol,UTF8]{ctex}.
```

```
Return the path to the generated PDF. If the translation content is
extensive, enable multiple subagents for parallel translation."
```

Then you can wait for the translation to complete. During this process, kimi will automatically analyze the directory structure, identify files to translate, and after translation, will attempt to compile. Based on any compilation errors, it will further adjust the translation until it successfully compiles into a PDF.

If you'd like to skip even downloading the source manually, you can try:

```
kimi -print -prompt "Download the paper source from arXiv with the
ID 2502.16982, extract it, and translate it into Chinese.  All English
text should be translated into Chinese, while formulas remain unchanged.
Translate only necessary captions for tables and figures, and only
translate necessary comments in code blocks or algorithm boxes -- do
not translate names.  Save the translated source into a new directory
named 'paper_cn' within the extraction directory, ensuring the source
remains compilable.  After translation, carefully check for any untranslated
sections or files.  Finally, recompile the translated result into a
PDF using xelatex.  If you encounter Chinese encoding issues during
compilation, consider using usepackage[fontset=fandol,UTF8]{ctex}.
Return the path to the generated PDF. If the translation content is
extensive, enable multiple subagents for parallel translation."
```

This example uses kimi-cli not solely as an advertisement, but because the author has only used kimi-cli — it was more of a "by the way" test during work rather than an intentional pursuit of AI Agent assistance, highlighting the author's "old-fashioned" nature. Readers are encouraged to try other AI agents they are comfortable with.

2 Translation Quality Comparison

Among existing paper translation websites, the author believes the best is Huan Jue Translation (hjfy.top). Below are the translation results of two papers using kimi and hjfy respectively:

1. kda_chinese_kimi.pdf;
2. kda_chinese_hjfy.pdf;
3. moonlight_chinese_kimi.pdf;
4. moonlight_chinese_hjfy.pdf.

You are encouraged to compare them yourself:

https://hjfy.top/arxiv/2510.26692

Kimi Linear：一种富有表现力、高效的注意力架构

Kimi Linear 技术报告

Kimi 团队

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

Abstract

我们介绍了 Kimi Linear，一种混合线性注意力架构，它首次在公平比较下于各种场景中超越了全注意力机制——包括短上下文，长上下文和强化学习（RL）扩展机制。其核心是 Kimi Delta Attention（KDA），这是一种富有表现力的线性注意力模块，它通过更细粒度的门控机制扩展了 Gated DeltaNet [111]，从而更有效地利用有限的有限状态 RNN 内存。我们定制的分块算法通过对角加低秩（DPLR）转移矩阵的专用变体实现了高硬件效率，与通用 DPLR 公式相比大幅减少了计算量，同时与经典 delta 规则保持更高的一致性。

我们预训练了一个具有 30 亿激活参数和 480 亿总参数的 Kimi Linear 模型，基于 KDA 和多头潜在注意力（MLA）的逐层混合架构，我们的实验表明，在相同的训练配方下，Kimi Linear 在所有评估任务上均以显著优势超越了全 MLA，同时将 KV 缓存使用量减少多达 75%，并在 1M 上下文长度下实现高达 6× 的解码吞吐量。这些结果表明，Kimi Linear 可以作为全注意力架构的即插即用替代品，具有更优越的性能和效率，包括处理更长输入和输出长度的任务。

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

1 引言

随着大型语言模型（LLM）逐渐演变为能力越来越强的智能体 [50]，推理的计算需求——特别是在长程和强化学习（RL）设置中——正在成为核心瓶颈。这种向 *RL 测试时扩展* [95, 33, 80, 74, 53] 的转变，其中模型必须在推理时处理扩展轨迹、工具使用交互和复杂决策空间，暴露了标准注意力机制的根本低效，特别是，softmax 注意力的二次时间复杂度和线性增长的键值（KV）缓存引入了巨大的计算和内存开销，阻碍了吞吐量、上下文长度扩展和实时交互性。

线性注意力 [48] 提供了一种降低计算复杂度的原则性方法，但由于表达能力有限，历史上在语言建模方面表现不如 softmax 注意力——即使对于短序列也是如此。最近的进展显著缩小了这一差距，主要通过两项创新：门控或衰减机制 [92, 16, 114] 和 delta 规则 [84, 112, 111, 71]。这些发展共同推动线性

[1] https://github.com/fla-org/flash-linear-attention/tree/main/fla/ops/kda

[2] https://huggingface.co/moonshotai/Kimi-Linear-48B-A3B-Instruct

Figure 1: kda__chinese__kimi

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**, **li2025survey**] 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

Figure 2: kda__chinese__hjfy

https://hjfy.top/arxiv/2502.16982

Muon 在大语言模型训练中的可扩展性研究

技术报告

Jingyuan Liu[1] **Jianlin Su**[1] **Xingcheng Yao**[2] **Zhejun Jiang**[1] **Guokun Lai**[1] **Yulun Du**[1] **Yidao Qin**[1] **Weixin Xu**[1] **Enzhe Lu**[1] **Junjie Yan**[1] **Yanru Chen**[1] **Huabin Zheng**[1] **Yibo Liu**[1] **Shaowei Liu**[1] **Bohong Yin**[1] **Weiran He**[1] **Han Zhu**[1] **Yuzhi Wang**[1] **Jianzhou Wang**[1] **Mengnan Dong**[1] **Zheng Zhang**[1] **Yongsheng Kang**[1] **Hao Zhang**[1] **Xinran Xu**[1] **Yutao Zhang**[1] **Yuxin Wu**[1] **Xinyu Zhou**[1]* **Zhilin Yang**[1]

[1] Moonshot AI [2] UCLA

Abstract

近期，基于矩阵正交化的 Muon 优化器 (K. Jordan et al. 2024) 在训练小规模语言模型方面展现出优异效果，但其在更大规模模型上的可扩展性尚未得到验证。我们识别出两项对 Muon 规模化扩展至关重要的技术：(1) 引入权重衰减 (weight decay) 和 (2) 精细调整逐参数更新尺度。这些技术使 Muon 能够在大规模训练中即插即用，无需进行超参数调优。扩展定律实验表明，在计算最优训练条件下，Muon 相比 AdamW 实现了约 ∼2× 的计算效率提升。基于这些改进，我们推出了 Moonlight，一个使用 Muon 训练、具有 3B/16B 参数的混合专家（MoE）模型，训练数据量为 5.7T 个 token。我们的模型提升了当前的帕累托前沿，相比先前模型以更少的训练 FLOPs 实现了更优的性能。我们开源了分布式 Muon 实现，该实现具有内存最优和通信高效的特点，我们还发布了预训练模型、指令微调模型以及中间训练检查点，以支持未来研究。

1 引言

大语言模型（LLMs）（OpenAI et al. 2024; DeepSeek-AI et al. 2024; Grattafiori et al. 2024; Gemini Team et al. 2024）的快速发展极大地推动了通用人工智能的进步。然而，由于扩展定律 (Kaplan et al. 2020; Hoffmann et al. 2022) 的存在，训练具有竞争力的大语言模型仍然是一个计算密集且资源需求巨大的过程。优化器在高效且有效地训练大语言模型方面发挥着关键作用，其中 Adam (Kingma et al. 2015) 及其变体 AdamW (Loshchilov et al. 2019) 是大多数大规模训练的标准选择。

近期优化算法的发展显示出超越 AdamW 提升训练效率的潜力 (Liu et al. 2024; K. Jordan et al. 2024; Yuan et al. 2024; Vyas et al. 2025; X.-L. Li 2018a; X.-L. Li 2018b; Pooladzandi et al. 2024; X. Li 2022; X.-L. Li 2024; Pethick et al. 2025)。其中，K. Jordan et al. 2024提出了 Muon，该优化器使用 Newton-Schulz 迭代对梯度动量进行正交化来更新矩阵参数。Muon 在小规模语言模型训练中的初步实

*通讯作者: zhouxinyu@moonshot.cn

Figure 3: moonlight__chinese__kimi

Muon 可扩展用于大语言模型训练

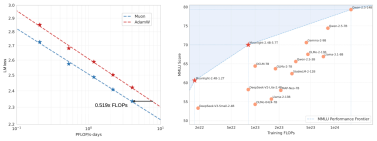
Technical Report

Jingyuan Liu[1] Jianlin Su[1] Xingcheng Yao[2] Zhejun Jiang[1] Guokun Lai[1] Yulun Du[1] Yidao Qin[1] Weixin Xu[1] Enzhe Lu[1] Junjie Yan[1] Yanru Chen[1] Huabin Zheng[1] Yibo Liu[1] Shaowei Liu[1] Bohong Yin[1] Weiran He[1] Han Zhu[1] Yuzhi Wang[1] Jianzhou Wang[1] Mengnan Dong[1] Zheng Zhang[1] Yongsheng Kang[1] Hao Zhang[1] Xinran Xu[1] Yutao Zhang[1] Yuxin Wu[1] Xinyu Zhou[1]* Zhilin Yang[1]

[1] Moonshot AI [2] UCLA

Abstract

最近，基于矩阵正交化的 Muon 优化器 (jordan2024muon) 在训练小规模语言模型方面展示了强劲的性能，但其在更大规模模型上的可扩展性尚未得到验证。我们确定了两个对 Muon 进行扩展的关键技术：(1) 添加权重衰减和 (2) 精心调整每个参数的更新比例。这些技术使得 Muon 能够无需调参即可在大规模训练中开箱即用，增强法用实验表明，与 AdamW 相比，Muon 在计算最优训练下实现了 ∼2× 的计算效率。基于这些改进，我们推出了 Moonlight，这是一个使用 Muon 训练，包含 3B/16B 参数的 MoE 模型，使用了 5.7T token 进行训练。我们的模型提升了当前的帕累托前沿，与先前的模型相比，在显著减少训练 FLOPs 的同时实现了更好的性能，我们开源了内存优化且通信高效的分布式 Muon 实现。同时，我们发布了预训练、指令调优及中间检查点，以支持未来的研究。



(a) (b)

图 1: 使用 Muon 进行扩展。(a) 比较 Muon 和 Adam 的缩放法则实验。在计算最优训练的情况下，Muon 的计算效率比 Adam 高 ∼ 2× 。(b) 我们用 Muon 优化的 Moonlight 模型以及其他可比模型的 MMLU 性能。Moonlight 推进了性能与训练 FLOP 的帕累托前沿。

*Corresponding author: zhouxinyu@moonshot.cn

Figure 4: moonlight__chinese__hjfy

3 A Few Additional Thoughts

Regarding the principles behind Huan Jue Translation, the author shared on his Zhihu article titled *"Spent 300 Million Tokens, I Used Large Models to Translate 10,000 arXiv Papers"*. Essentially, the idea is quite straightforward: starting from the LaTeX source (which is generally available for arXiv papers), extract the paragraphs requiring translation and translate them one by one.

However, when actually attempting this, one quickly feels "at a loss" — because LaTeX is an extremely flexible syntax, to the point where it's not just a document syntax anymore but almost a programming tool. In this context, trying to manually break down the "extraction-translation" process into a finite number of steps becomes extremely difficult. As the author notes, he still needs to manually fix translation errors (bad cases) daily, which is a long, tedious, and never-ending task.

This is exactly where AI Agents shine — tasks that are difficult to manually decompose into enumerable steps. The AI automatically identifies and translates the necessary files. Even better, it automatically compiles the document and adjusts based on error messages until the compilation succeeds. If you were to write rules manually, just handling the compilation error messages alone could become an endless task, with the essential difficulty possibly being no less than rewriting a LaTeX compiler — and the difficulties involved are.

Besides automation, another advantage of using kimi-cli is translation quality. Compared to translating paragraph-by-paragraph or sentence-by-sentence, kimi-cli benefits from global contextual information, resulting in significantly better overall translation quality. Of course, there are still issues — the typical "hallucination" problem with large models. Specifically, translation errors may include omissions or incorrect translations. For omitted content, in an interactive environment you can repeatedly request kimi to check for missed sections until satisfied. However, for incorrect translations, there's not much that can be done.

4 Summary

This article presents a basic usage of kimi-cli for translating arXiv papers. Readers are welcome to share more comprehensive usage examples.