The Big LLM Architecture Comparison
大语言模型架构比较

From DeepSeek-V3 to Kimi K2: A Look At Modern LLM Architecture Design

从DeepSeek-V3到雀灵K2：现代大语言模型架构设计剖析

SEBASTIAN RASCHKA, PHD 塞巴斯蒂安·拉施卡博士

JUL 19, 2025 2025年7月19日

It has been seven years since the original GPT architecture was developed. A first glance, looking back at GPT-2 (2019) and forward to DeepSeek-V3 and Llama 4 (2024-2025), one might be surprised at how structurally similar thes models still are.

自最初的GPT架构开发以来，已经过去了七年。乍一看，回顾GPT-2（2019年），望DeepSeek-V3和Llama 4（2024 - 2025年），人们可能会惊讶于这些模型在结构仍然如此相似。

Sure, positional embeddings have evolved from absolute to rotational (RoPE Multi-Head Attention has largely given way to Grouped-Query Attention, an the more efficient SwiGLU has replaced activation functions like GELU. But beneath these minor refinements, have we truly seen groundbreaking chang or are we simply polishing the same architectural foundations?

当然，位置嵌入已经从绝对位置嵌入发展到旋转位置嵌入（RoPE），多头注意力在大程度上已被分组查询注意力所取代，更高效的SwiGLU激活函数也已取代了诸如G之类的激活函数。但在这些细微改进的背后，我们是否真的看到了突破性的变化，还仅仅在对相同的架构基础进行优化？

Comparing LLMs to determine the key ingredients that contribute to their g (or not-so-good) performance is notoriously challenging: datasets, training techniques, and hyperparameters vary widely and are often not well

documented.

比较大语言模型（LLMs）以确定影响其性能优劣的关键因素极具挑战性：数据集、练技术和超参数差异很大，且往往没有详细记录。

However, I think that there is still a lot of value in examining the structural changes of the architectures themselves to see what LLM developers are up in 2025. (A subset of them are shown in Figure 1 below.)

然而，我认为研究架构本身的结构变化，以了解2025年大语言模型开发者在做什么仍然有很大价值。（其中一部分展示在下面的图1中。）

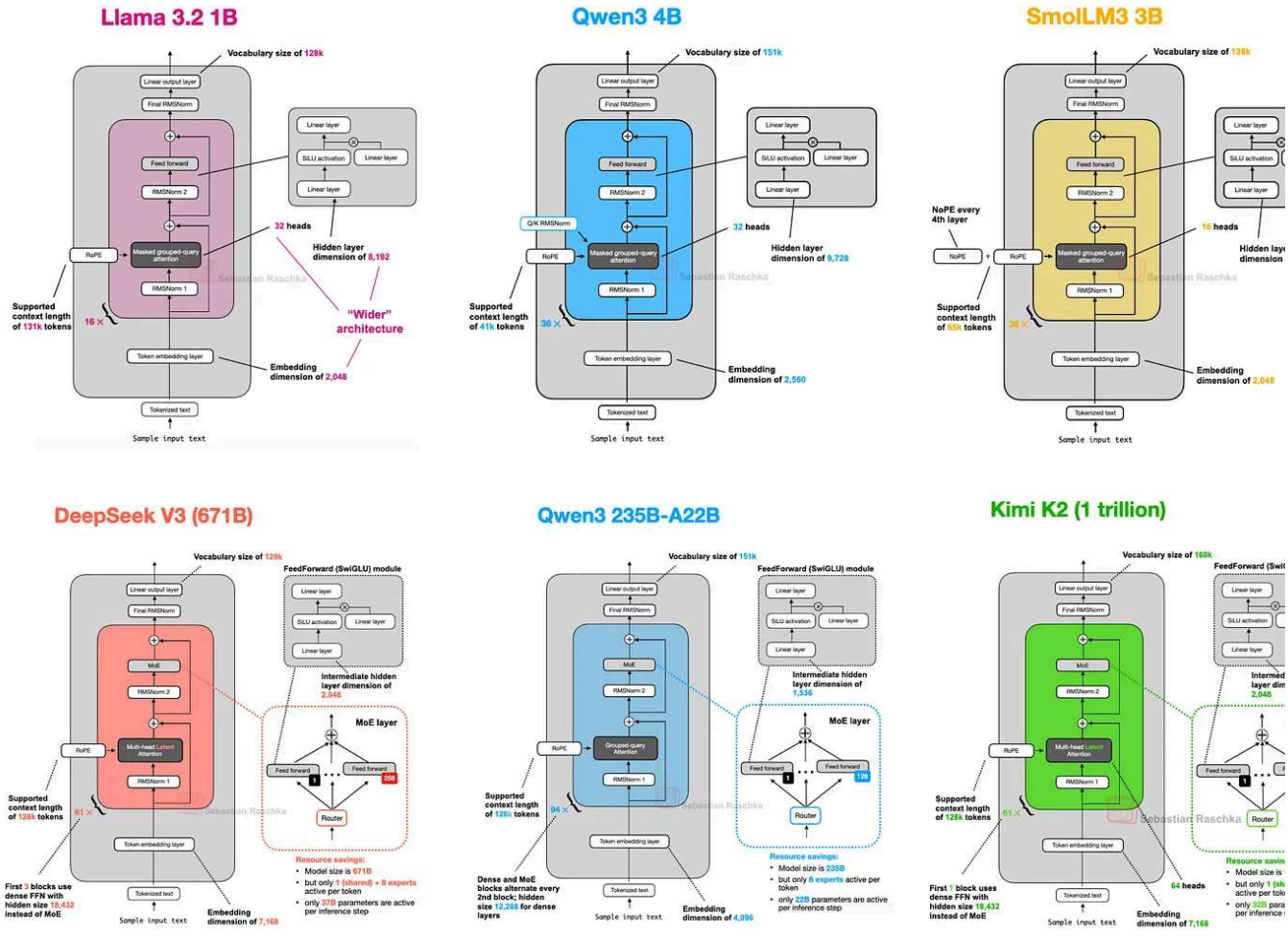


Figure 1: A subset of the architectures covered in this article.

图1：本文所涵盖架构的一个子集。

So, in this article, rather than writing about benchmark performance or trair algorithms, I will focus on the architectural developments that define today'

flagship open models.

所以，在这篇文章中，我不会去写基准性能或训练算法，而是将重点放在那些定义今旗舰级开源模型的架构发展上。

(As you may remember, I wrote about multimodal LLMs not too long ago; in this article, I will focus on the text capabilities of recent models and leave the discussion of multimodal capabilities for another time.)

（你们可能还记得，不久前我写过关于多模态大语言模型的文章；在本文中，我将于近期模型的文本能力，把多模态能力的讨论留到其他时间。）

Tip: This is a fairly comprehensive article, so I recommend using the navigat bar to access the table of contents (just hover over the left side of the Subs page).

提示： 这是一篇相当全面的文章，因此我建议使用导航栏来访问目录（只需将鼠标在Substack页面的左侧）。

1. DeepSeek V3/R1 1. 深度求索V3/R1

As you have probably heard more than once by now, DeepSeek R1 made a impact when it was released in January 2025. DeepSeek R1 is a reasoning model built on top of the DeepSeek V3 architecture, which was introduced i December 2024.

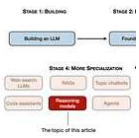
如你现在可能已经不止一次听说过的，DeepSeek R1在2025年1月发布时产生了巨响。DeepSeek R1是一款基于DeepSeek V3架构构建的推理模型，该架构于2024月推出。

While my focus here is on architectures released in 2025, I think it's reasona to include DeepSeek V3, since it only gained widespread attention and adoption following the launch of DeepSeek R1 in 2025.

虽然我在这里关注的是2025年发布的架构，但我认为将DeepSeek V3包括在内是合的，因为它在2025年DeepSeek R1推出后才获得广泛关注和应用。

If you are interested in the training of DeepSeek R1 specifically, you may als find my article from earlier this year useful:

如果您特别关注DeepSeek R1的训练，您可能也会发现我今年早些时候的文章很有用：



Understanding Reasoning LLMs 理解推理大语言模型

SEBASTIAN RASCHKA, PHD 塞巴斯蒂安·拉施卡博士 · 2月5日

Read full story 阅读完整故事 →

In this section, I'll focus on two key architectural techniques introduced in DeepSeek V3 that improved its computational efficiency and distinguish it f many other LLMs:

在本节中，我将重点介绍DeepSeek V3中引入的两项关键架构技术，这些技术提高其计算效率，并使其有别于许多其他大语言模型：

- Multi-Head Latent Attention (MLA) 多头潜在注意力（MLA）
- Mixture-of-Experts (MoE) 混合专家（MoE）

1.1 Multi-Head Latent Attention (MLA)
1.1 多头潜在注意力（MLA）

Before discussing Multi-Head Latent Attention (MLA), let's briefly go over sc background to motivate why it's used. For that, let's start with Grouped-Que Attention (GQA), which has become the new standard replacement for a mc compute- and parameter-efficient alternative to Multi-Head Attention (MHA recent years.

在讨论多头潜在注意力（MLA）之前，让我们简要回顾一下一些背景知识，以说明么要使用它。为此，让我们从分组查询注意力（GQA）开始，近年来，它已成为多注意力（MHA）在计算和参数效率方面更优的新标准替代方案。

So, here's a brief GQA summary. Unlike MHA, where each head also has its c set of keys and values, to reduce memory usage, GQA groups multiple heac

share the same key and value projections.

所以，这里简要总结一下GQA。与多头注意力机制（MHA）不同，在MHA中每个有自己的一组键值对，而GQA为了减少内存使用，将多个头分组以共享相同的键值影。

For example, as further illustrated in Figure 2 below, if there are 2 key-value groups and 4 attention heads, then heads 1 and 2 might share one set of ke and values, while heads 3 and 4 share another. This reduces the total numbe key and value computations, which leads to lower memory usage and impro efficiency (without noticeably affecting the modeling performance, accordir ablation studies).

例如，如下图2进一步所示，如果有2个键值组和4个注意力头，那么头1和头2可能一组键值，而头3和头4共享另一组。这减少了键值计算的总数，从而降低了内存使提高了效率（根据消融研究，不会显著影响建模性能）。

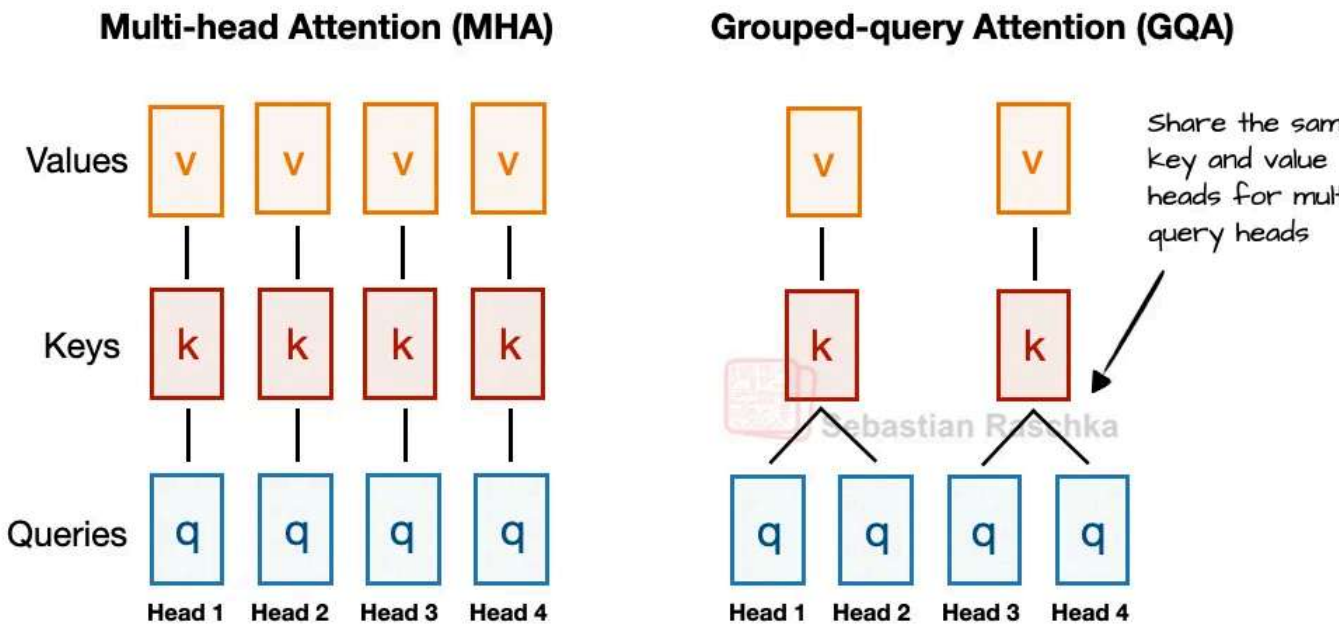


Figure 2: A comparison between MHA and GQA. Here, the group size is 2, where a key and value pair is shared among 2 queries.

图2：多头注意力机制（MHA）与组查询注意力机制（GQA）的比较。此处，组大小为2，即一个键值对由2个查询共享。

So, the core idea behind GQA is to reduce the number of key and value hea by sharing them across multiple query heads. This (1) lowers the model's parameter count and (2) reduces the memory bandwidth usage for key and

value tensors during inference since fewer keys and values need to be store and retrieved from the KV cache.

因此，GQA背后的核心思想是通过在多个查询头之间共享键头和值头，来减少键头头的数量。这一做法（1）减少了模型的参数数量，并且（2）由于推理过程中需要值缓存中存储和检索的键和值更少，从而降低了键张量和值张量的内存带宽使用量

(If you are curious how GQA looks in code, see my GPT-2 to Llama 3 conver guide for a version without KV cache and my KV-cache variant here.)

（如果你好奇GQA在代码中是什么样子，可查看我的GPT-2转Llama 3转换指南，有一个无KV缓存的版本，而我带KV缓存的变体版本见 此处。）

While GQA is mainly a computational-efficiency workaround for MHA, ablat studies (such as those in the original GQA paper and the Llama 2 paper) sho performs comparably to standard MHA in terms of LLM modeling performa

虽然全局查询注意力（GQA）主要是针对多头注意力（MHA）的一种计算效率解决案，但消融研究（如原始GQA论文和 Llama 2论文中的研究）表明，在大语言模型性能方面，它的表现与标准的多头注意力（MHA）相当。

Now, Multi-Head Latent Attention (MLA) offers a different memory-saving strategy that also pairs particularly well with KV caching. Instead of sharing and value heads like GQA, MLA compresses the key and value tensors into a lower-dimensional space before storing them in the KV cache.

如今，多头潜在注意力（MLA）提供了一种不同的节省内存策略，该策略与键值缓（KV caching）也特别匹配。与分组查询注意力（GQA）不同，MLA不会共享键头值头，而是在将键张量和值张量存储到键值缓存中之前，将它们压缩到一个低维空

At inference time, these compressed tensors are projected back to their orig size before being used, as shown in the Figure 3 below. This adds an extra matrix multiplication but reduces memory usage.

在推理时，如图3所示，这些压缩后的张量在使用前会被投影回原始大小。这增加了次额外的矩阵乘法，但减少了内存使用。

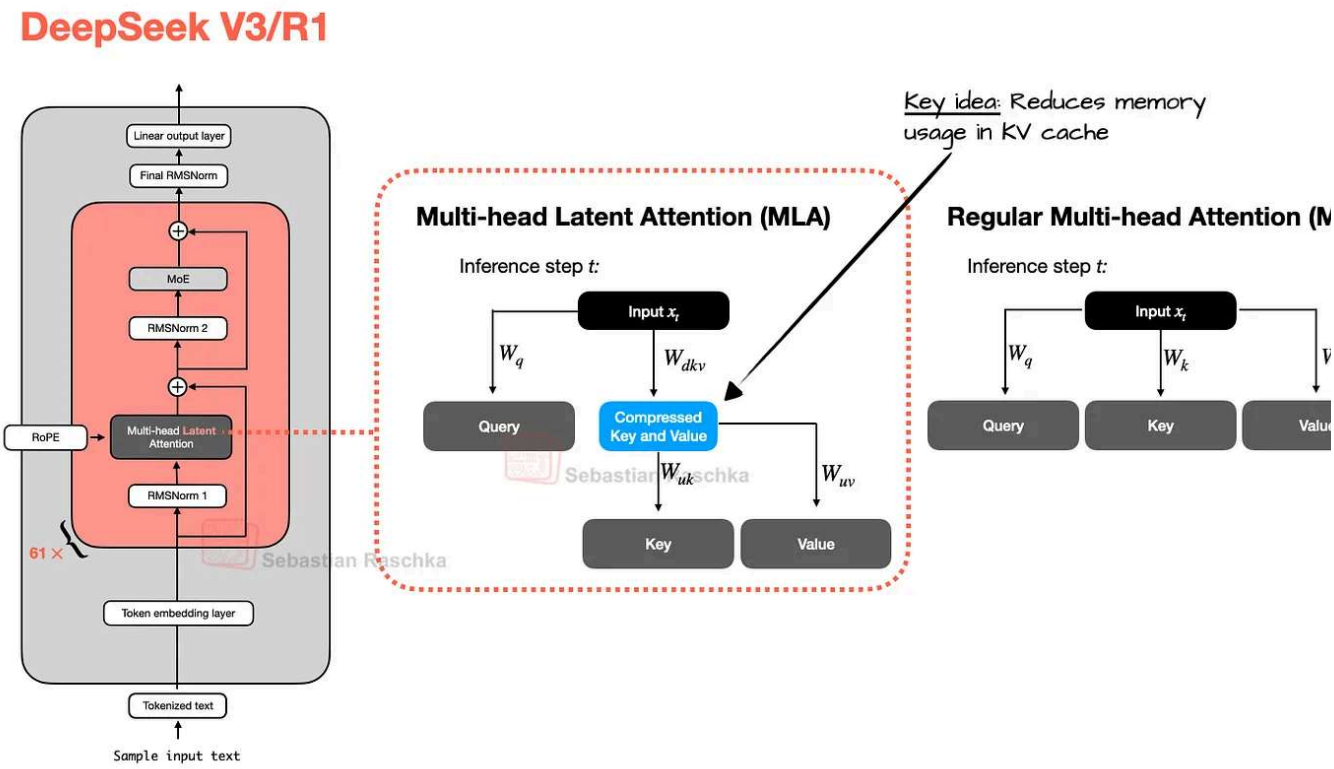


Figure 3: Comparison between MLA (used in DeepSeek V3 and R1) and regular MHA.

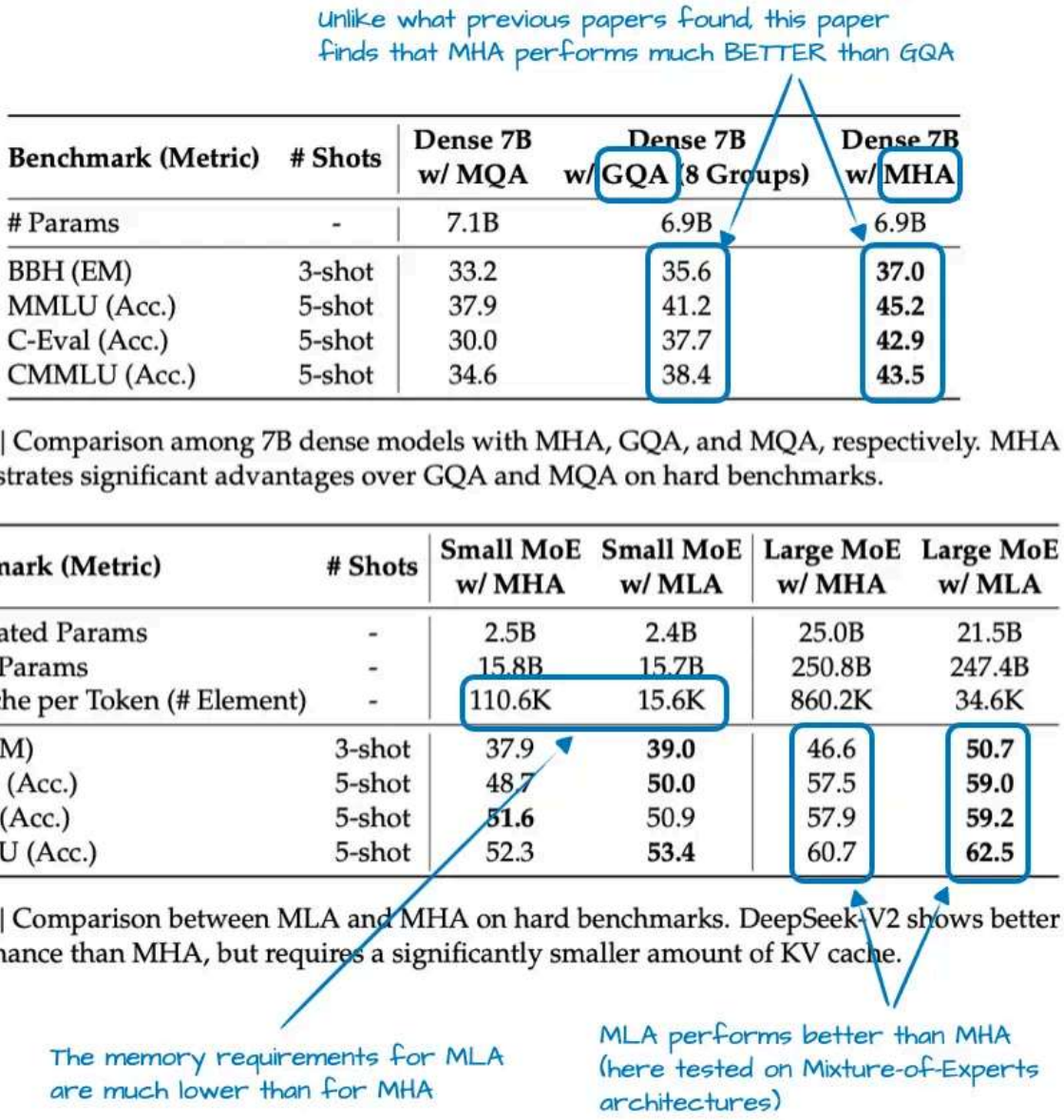
图3：MLA（用于DeepSeek V3和R1）与常规MHA的对比。

(As a side note, the queries are also compressed, but only during training, n inference.)

（顺便说一句，查询也会被压缩，但仅在训练期间，推理期间不会。）

By the way, MLA is not new in DeepSeek V3, as its DeepSeek-V2 predecessc also used (and even introduced) it. Also, the V2 paper contains a few interes ablation studies that may explain why the DeepSeek team chose MLA over (see Figure 4 below).

顺便说一下，MLA在DeepSeek V3中并非新事物，因为其前身DeepSeek - V2也使了（甚至是引入了）它。此外，V2的论文包含一些有趣的消融研究，这些研究或许释为什么DeepSeek团队选择MLA而非GQA（见下图4）。



Benchmark (Metric)	# Shots	Dense 7B w/ MQA	Dense 7B w/ GQA (8 Groups)	Dense 7B w/ MHA
# Params	-	7.1B	6.9B	6.9B
BBH (EM)	3-shot	33.2	35.6	**37.0**
MMLU (Acc.)	5-shot	37.9	41.2	**45.2**
C-Eval (Acc.)	5-shot	30.0	37.7	**42.9**
CMMLU (Acc.)	5-shot	34.6	38.4	**43.5**

Table 8 | Comparison among 7B dense models with MHA, GQA, and MQA, respectively. MHA demonstrates significant advantages over GQA and MQA on hard benchmarks.

Benchmark (Metric)	# Shots	Small MoE w/ MHA	Small MoE w/ MLA	Large MoE w/ MHA	Large MoE w/ MLA
# Activated Params	-	2.5B	2.4B	25.0B	21.5B
# Total Params	-	15.8B	15.7B	250.8B	247.4B
KV Cache per Token (# Element)	-	110.6K	15.6K	860.2K	34.6K
BBH (EM)	3-shot	37.9	**39.0**	46.6	**50.7**
MMLU (Acc.)	5-shot	48.7	**50.0**	57.5	**59.0**
C-Eval (Acc.)	5-shot	**51.6**	50.9	57.9	**59.2**
CMMLU (Acc.)	5-shot	52.3	**53.4**	60.7	**62.5**

Table 9 | Comparison between MLA and MHA on hard benchmarks. DeepSeek-V2 shows better performance than MHA, but requires a significantly smaller amount of KV cache.

Figure 4: Annotated tables from the DeepSeek-V2 paper, https://arxiv.org/abs/2405.04434

图4：来自DeepSeek-V2论文的注释表格，https://arxiv.org/abs/2405.04434

As shown in Figure 4 above, GQA appears to perform worse than MHA, whereas MLA offers better modeling performance than MHA, which is likely why the DeepSeek team chose MLA over GQA. (It would have been interest to see the "KV Cache per Token" savings comparison between MLA and GQ well!)

如图4所示，GQA的表现似乎比多头注意力机制（MHA）差，而MLA的建模性能似 MHA，这可能就是DeepSeek团队选择MLA而非GQA的原因。（要是能看到MLA GQA在“每个令牌的键值缓存”节省方面的比较，那就更有意思了！）

To summarize this section before we move on to the next architecture component, MLA is a clever trick to reduce KV cache memory use while eve slightly outperforming MHA in terms of modeling performance.

在进入下一个架构组件之前，总结一下本节内容：多位置注意力（MLA）是一种巧方法，它在减少键值（KV）缓存内存使用的同时，在建模性能方面甚至略优于多头力（MHA）。

1.2 Mixture-of-Experts (MoE)
1.2 混合专家（MoE）

The other major architectural component in DeepSeek worth highlighting is use of Mixture-of-Experts (MoE) layers. While DeepSeek did not invent MoE has seen a resurgence this year, and many of the architectures we will cover later also adopt it.

DeepSeek中另一个值得强调的主要架构组件是其对专家混合（MoE）层的使用。 DeepSeek并非MoE的发明者，但MoE在今年重新兴起，我们稍后将介绍的许多架采用了这一技术。

You are likely already familiar with MoE, but a quick recap may be helpful.

你可能已经对混合专家（MoE）很熟悉了，但快速回顾一下可能会有所帮助。

The core idea in MoE is to replace each FeedForward module in a transform block with multiple expert layers, where each of these expert layers is also a FeedForward module. This means that we swap a single FeedForward block multiple FeedForward blocks, as illustrated in the Figure 5 below.

混合专家（MoE）的核心思想是，用多个专家层取代Transformer模块中的每个前 块，其中每个专家层也是一个前馈模块。这意味着我们将单个前馈模块替换为多个 模块，如下图5所示。

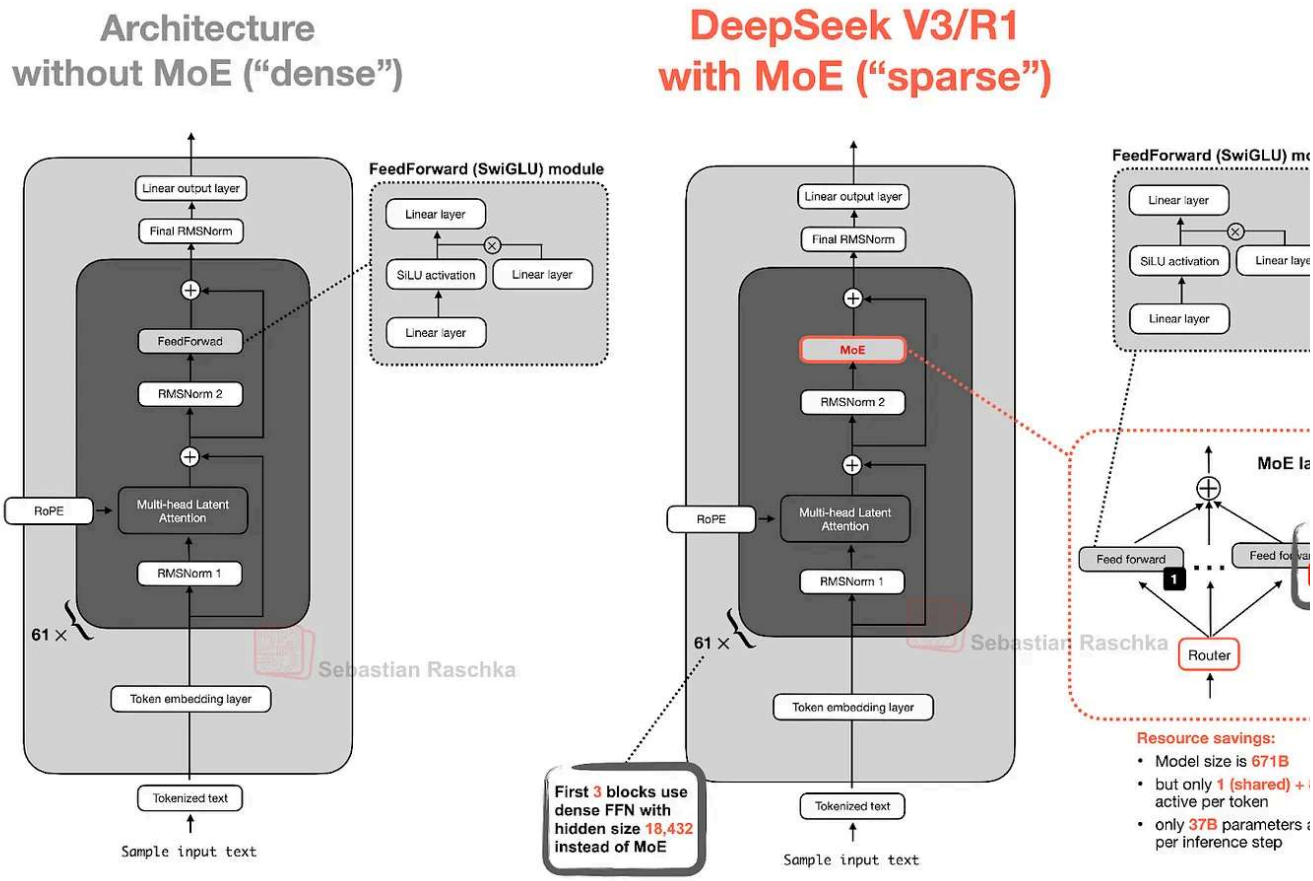


Figure 5: An illustration of the Mixture-of-Experts (MoE) module in DeepSeek V3/R1 (right) compared to an LLM with a standard FeedForward block (left).

图5：与具有标准前馈模块的大语言模型（左）相比，DeepSeek V3/R1中专家混合（MoE）模块的示意图（右）。

The FeedForward block inside a transformer block (shown as the dark gray block in the figure above) typically contains a large number of the model's t parameters. (Note that the transformer block, and thereby the FeedForwarc block, is repeated many times in an LLM; in the case of DeepSeek-V3, 61 tin

Transformer模块中的前馈网络模块（在上图中显示为深灰色模块）通常包含模型总数中的很大一部分。（请注意，Transformer模块，进而前馈网络模块，在大语言模中会重复多次；以DeepSeek-V3为例，重复了61次。）

So, replacing *a single* FeedForward block with *multiple* FeedForward blocks done in a MoE setup) substantially increases the model's total parameter co However, the key trick is that we don't use ("activate") all experts for every token. Instead, a router selects only a small subset of experts per token. (In interest of time, or rather article space, I'll cover the router in more detail

another time.)

因此，用*多个*前馈块替换*单个*前馈块（如在混合专家（MoE）设置中所做的那样）幅增加模型的总参数数量。然而，关键技巧在于，我们并非对每个词元都使用（“激活”）所有专家。相反，一个路由选择器为每个词元仅选择一小部分专家。（考虑到间，或者更确切地说是文章篇幅，我将在其他时间更详细地介绍路由选择器。）

Because only a few experts are active at a time, MoE modules are often refe to as *sparse*, in contrast to *dense* modules that always use the full paramete set. However, the large total number of parameters via an MoE increases the capacity of the LLM, which means it can take up more knowledge during training. The sparsity keeps inference efficient, though, as we don't use all tl parameters at the same time.

由于一次只有少数专家处于激活状态，混合专家（MoE）模块通常被称为*稀疏*模块与始终使用全部参数集的*密集*模块形成对比。然而，通过混合专家（MoE）获得的参数总数增加了大语言模型（LLM）的能力，这意味着它在训练过程中可以吸收更识。不过，稀疏性使推理保持高效，因为我们不会同时使用所有参数。

For example, DeepSeek-V3 has 256 experts per MoE module and a total of billion parameters. Yet during inference, only 9 experts are active at a time (shared expert plus 8 selected by the router). This means just 37 billion parameters are used per inference step as opposed to all 671 billion.

例如，DeepSeek-V3每个混合专家（MoE）模块有256个专家，总共有6710亿个参数。然而在推理过程中，每次只有9个专家处于激活状态（1个共享专家加上由路由择的8个专家）。这意味着每次推理步骤仅使用370亿个参数，而不是全部6710亿个数。

One notable feature of DeepSeek-V3's MoE design is the use of a shared ex This is an expert that is always active for every token. This idea is not new a was already introduced in the DeepSeek 2024 MoE and 2022 DeepSpeedMo papers.

DeepSeek-V3的混合专家（MoE）设计的一个显著特点是使用共享专家。这是一个每个词元始终处于激活状态的专家。这个想法并不新鲜，在DeepSeek 2024 MoE和2022年DeepSpeedMoE论文中就已经提出。

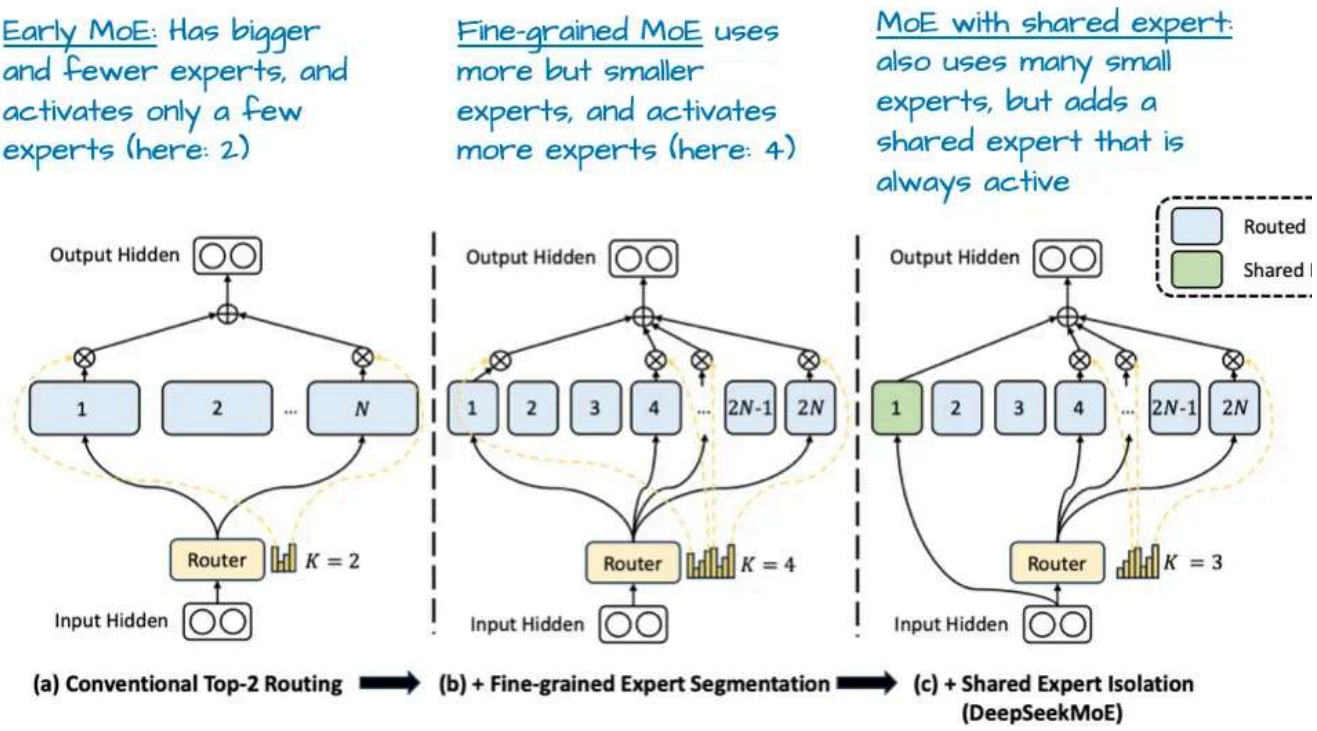


Figure 6: An annotated figure from "DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models", https://arxiv.org/abs/2401.06066

图6：来自《DeepSeekMoE：迈向混合专家语言模型的终极专家专业化》（https://arxiv.org/abs/2401.06066）的注释图

The benefit of having a shared expert was first noted in the DeepSpeedMoE paper, where they found that it boosts overall modeling performance comp to no shared experts. This is likely because common or repeated patterns d have to be learned by multiple individual experts, which leaves them with m room for learning more specialized patterns.

在《DeepSpeedMoE论文》中首次提到了设置共享专家的好处，文中发现，与没有 享专家相比，这提升了整体建模性能。这很可能是因为常见或重复的模式无需由多 立专家分别学习，从而为专家留出了更多空间来学习更专业的模式。

1.3 DeepSeek Summary 1.3 深度求索总结

To summarize, DeepSeek-V3 is a massive 671-billion-parameter model that, launch, outperformed other open-weight models, including the 405B Llama Despite being larger, it is much more efficient at inference time thanks to its Mixture-of-Experts (MoE) architecture, which activates only a small subset o

(just 37B) parameters per token.

总而言之，DeepSeek-V3是一款拥有6710亿参数的大规模模型，发布时其性能超越
包括4050亿参数的Llama 3在内的其他开源权重模型。尽管参数规模更大，但由于
了混合专家（MoE）架构，它在推理时的效率要高得多，该架构每次处理一个词元
激活一小部分（仅370亿）参数。

Another key distinguishing feature is DeepSeek-V3's use of Multi-Head Late
Attention (MLA) instead of Grouped-Query Attention (GQA). Both MLA and
GQA are inference-efficient alternatives to standard Multi-Head Attention
(MHA), particularly when using KV caching. While MLA is more complex to
implement, a study in the DeepSeek-V2 paper has shown it delivers better
modeling performance than GQA.

另一个关键的区别特征是，DeepSeek-V3使用多头潜在注意力（MLA）而非分组查
注意力（GQA）。MLA和GQA都是标准多头注意力（MHA）在推理效率上的替代
案，尤其是在使用键值缓存（KV caching）时。虽然MLA的实现更为复杂，但
《DeepSeek-V2论文》中的一项研究表明，它的建模性能优于GQA。

2. OLMo 2

The OLMo series of models by the non-profit Allen Institute for AI is
noteworthy due to its transparency in terms of training data and code, as w
as the relatively detailed technical reports.

非营利组织艾伦人工智能研究所（Allen Institute for AI）的OLMo系列模型值得关
注，因为其训练数据和代码具有透明度，而且技术报告也相对详细。

While you probably won't find OLMo models at the top of any benchmark c
leaderboard, they are pretty clean and, more importantly, a great blueprint
developing LLMs, thanks to their transparency.

虽然你可能不会在任何基准测试或排行榜的榜首找到OLMo模型，但它们相当纯粹
重要的是，由于其透明度，它们是开发大语言模型（LLMs）的绝佳蓝本。

And while OLMo models are popular because of their transparency, they are
that bad either. In fact, at the time of release in January (before Llama 4,

Gemma 3, and Qwen 3), OLMo 2 models were sitting at the Pareto frontier c compute to performance, as shown in Figure 7 below.

虽然OLMo模型因其透明度而受到欢迎，但它们也并不差。事实上，在1月份发布时（在Llama 4、Gemma 3和通义千问3之前），OLMo 2模型处于计算性能的帕累托沿，如下方图7所示。

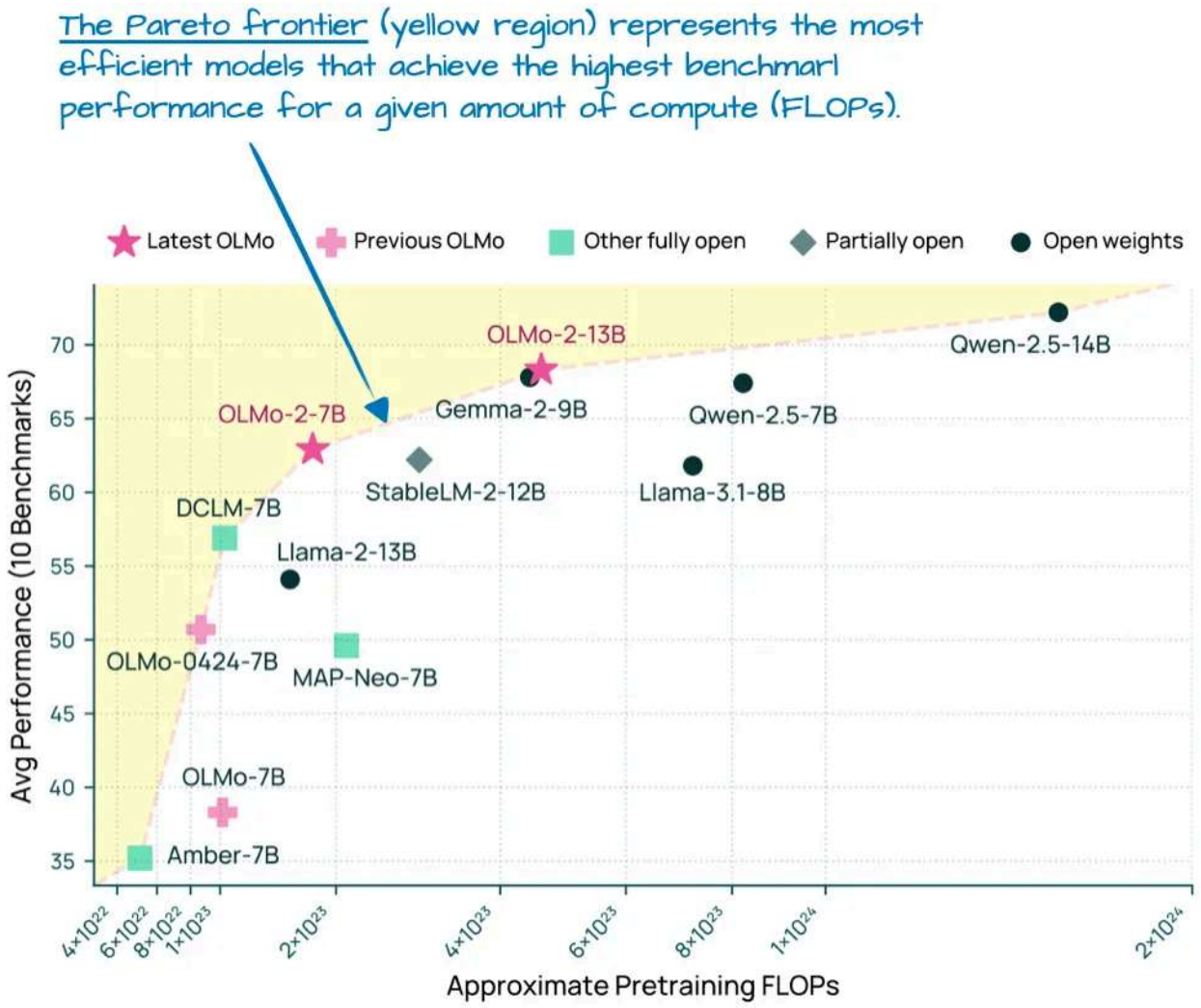


Figure 7: Modeling benchmark performance (higher is better) vs pre-training cost (FLOPs; lower is better) for different LLMs. This is an annotated figure from the OLMo 2 paper, https://arxiv.org/abs/2501.00656

图7：不同大语言模型的建模基准性能（越高越好）与预训练成本（每秒浮点运算次数；越低越好）对比。这是来自OLMo 2论文（https://arxiv.org/abs/2501.00656）的一张注释图。

As mentioned earlier in this article, I aim to focus only on the LLM architectu details (not training or data) to keep it at a manageable length. So, what we the interesting architectural design choices in OLMo2 ? It mainly comes dov to normalizations: the placement of RMSNorm layers as well as the additior

a QK-norm, which I will discuss below.

正如本文前面所提到的，为了使内容篇幅可控，我旨在仅聚焦于大语言模型（LLM
架构细节（而非训练或数据）。那么，OLMo2 中有哪些有趣的架构设计选择呢？
要归结于归一化处理：均方根归一化（RMSNorm）层的位置，以及增加的QK归一
化，下面我将对此展开讨论。

Another thing worth mentioning is that OLMo 2 still uses traditional Multi-H
Attention (MHA) instead of MLA or GQA.

另一件值得一提的事是，OLMo 2仍然使用传统的多头注意力机制（MHA），而非
MLA或GQA。

2.1 Normalization Layer Placement
2.1 归一化层的位置

Overall, OLMo 2 largely follows the architecture of the original GPT model, similar to other contemporary LLMs. However, there are some noteworthy deviations. Let's start with the normalization layers.

总体而言，OLMo 2在很大程度上沿用了原始GPT模型的架构，这与其他当代大语言
型类似。然而，也存在一些值得注意的差异。我们先从归一化层说起。

Similar to Llama, Gemma, and most other LLMs, OLMo 2 switched from LayerNorm to RMSNorm.

与Llama、Gemma以及大多数其他大语言模型类似，OLMo 2从层归一化
（LayerNorm）改为了均方根归一化（RMSNorm）。

But since RMSNorm is old hat (it's basically a simplified version of LayerNor with fewer trainable parameters), I will skip the discussion of RMSNorm vs LayerNorm. (Curious readers can find an RMSNorm code implementation ir GPT-2 to Llama conversion guide.)

但由于均方根归一化（RMSNorm）已是老生常谈（它基本上是层归一化（LayerNorm）的简化版本，可训练参数更少），我将跳过关于均方根归一化与层化对比的讨论。（好奇的读者可以在我的《GPT-2转Llama转换指南》中找到均方根一化的代码实现。）

However, it's worth discussing the placement of the RMSNorm layer. The original transformer (from the "Attention is all you need" paper) placed the normalization layers in the transformer block *after* the attention module and the FeedForward module, respectively.

然而，均方根归一化（RMSNorm）层的位置值得探讨。原始的Transformer（来自《“Attention is all you need”》论文）将两个归一化层分别置于Transformer模块意力模块和前馈模块*之后*。

This is also known as Post-LN or Post-Norm.

这也被称为后置层归一化（Post-LN）或后置归一化（Post-Norm）。

GPT and most other LLMs that came after placed the normalization layers *before* the attention and FeedForward modules, which is known as Pre-LN or Pre-Norm. A comparison between Post- and Pre-Norm is shown in the figure below.

GPT和随后出现的大多数其他大语言模型（LLMs）将归一化层置于注意力模块和前馈模块**之前**，这被称为前置层归一化（Pre-LN）或前置归一化（Pre-Norm）。后归一化和前置归一化的对比见下图。

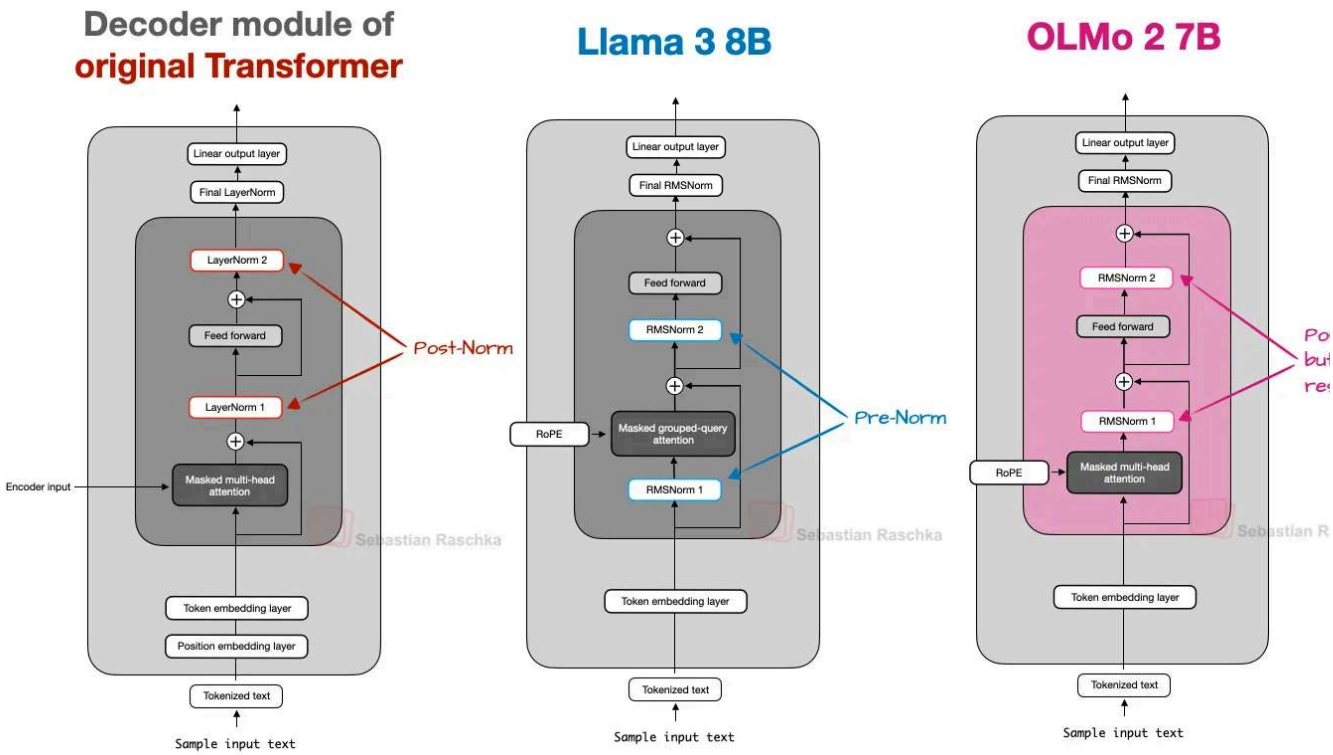


Figure 8: A comparison of Post-Norm, Pre-Norm, and OLMo 2's flavor of Post-Norm.

图8：后置归一化、前置归一化与OLMo 2的后置归一化方式对比。

In 2020, Xiong et al. showed that Pre-LN results in more well-behaved gradi at initialization. Furthermore, the researchers mentioned that Pre-LN even works well without careful learning rate warm-up, which is otherwise a cruc tool for Post-LN.

2020年，熊等人表明，前置归一化（Pre-LN）在初始化时能产生更易于处理的梯度 此外，研究人员提到，前置归一化（Pre-LN）即使在没有精心设置学习率预热的情 也能表现良好，而学习率预热对于后置归一化（Post-LN）来说却是一个关键手段。

Now, the reason I am mentioning that is that OLMo 2 adopted a form of Po LN (but with RMSNorm instead of LayerNorm, so I am calling it *Post-Norm*).

现在，我提到这一点的原因是，OLMo 2采用了一种后置归一化（Post-LN）的形式 （但使用均方根归一化（RMSNorm）而非层归一化（LayerNorm），所以我将其 *后置归一化（Post-Norm）*）。

In OLMo 2, instead of placing the normalization layers before the attention FeedForward layers, they place them after, as shown in the figure above. However, notice that in contrast to the original transformer architecture, the

normalization layers are still inside the residual layers (skip connections).

在OLMo 2中，与将归一化层置于注意力层和前馈层之前不同，他们将其置于之后，上图所示。然而，请注意，与原始Transformer架构不同，归一化层仍位于残差层跃连接）内部。

So, why did they move the position of the normalization layers? The reason that it helped with training stability, as shown in the figure below.

那么，他们为什么要移动归一化层的位置呢？ 原因是这样做有助于训练的稳定性，图所示。

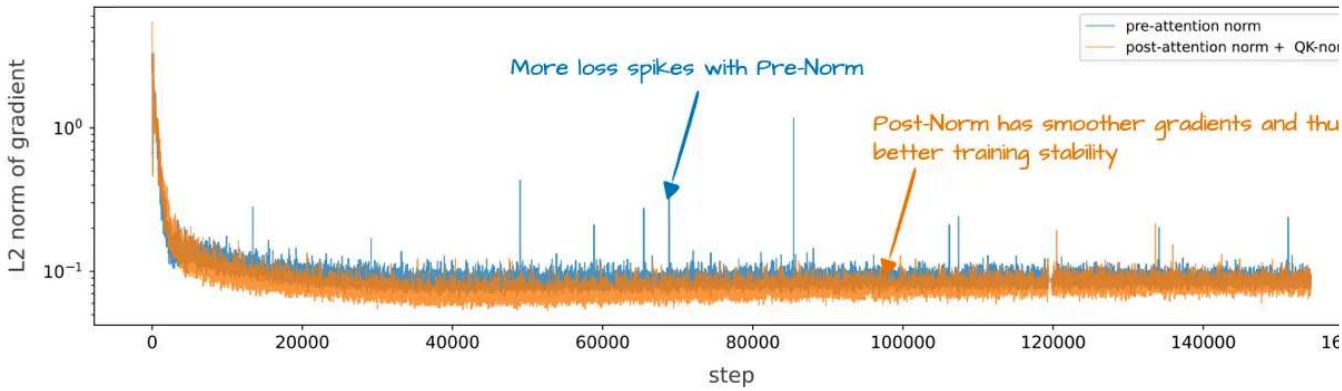


Figure 9: A plot showing the training stability for Pre-Norm (like in GPT-2, Llama 3, and many others) versus OLMo 2's flavor of Post-Norm.

图9：一张展示预归一化（如GPT-2、Llama 3及其他许多模型中所采用）与OLMo 2的后归一化变体在训练稳定性方面对比的图表。

Unfortunately this figure shows the results of the reordering together with Norm, which is a separate concept. So, it's hard to tell how much the normalization layer reordering contributed by itself.

不幸的是，该图展示了重排序与QK归一化共同作用的结果，而QK归一化是一个独立概念。因此，很难判断归一化层重排序本身贡献了多少。

2.2 QK-Norm 2.2 QK归一化

Since the previous section already mentioned the QK-norm, and other LLMs discuss later, such as Gemma 2 and Gemma 3, also use QK-norm, let's briefl

discuss what this is.

由于上一节已经提到了QK范数，而且我们稍后讨论的其他大语言模型，如Gemma
Gemma 3，也使用QK范数，下面我们简要讨论一下这是什么。

QK-Norm is essentially yet another RMSNorm layer. It's placed inside the M
Head Attention (MHA) module and applied to the queries (q) and keys (k)
before applying RoPE. To illustrate this, below is an excerpt of a Grouped-Q
Attention (GQA) layer I wrote for my Qwen3 from-scratch implementation (
QK-norm application in GQA is similar to MHA in OLMo):

QK归一化本质上是另一个RMS归一化层。它位于多头注意力（MHA）模块内部，
用旋转位置嵌入（RoPE）之前应用于查询（q）和键（k）。为了说明这一点，下面
我为从零开始实现的Qwen3编写的分组查询注意力（GQA）层的一段代码（GQA
QK归一化应用与OLMo中的MHA类似）：

```
class GroupedQueryAttention(nn.Module):
    def __init__(
        self, d_in, num_heads, num_kv_groups,
        head_dim=None, qk_norm=False, dtype=None
    ):
        # ...

        if qk_norm:
            self.q_norm = RMSNorm(head_dim, eps=1e-6)
            self.k_norm = RMSNorm(head_dim, eps=1e-6)
        else:
            self.q_norm = self.k_norm = None

    def forward(self, x, mask, cos, sin):
        b, num_tokens, _ = x.shape

        # Apply projections
        queries = self.W_query(x)
        keys = self.W_key(x)
        values = self.W_value(x)

        # ...

        # Optional normalization
        if self.q_norm:
```

```
        queries = self.q_norm(queries)
    if self.k_norm:
        keys = self.k_norm(keys)

    # Apply RoPE
    queries = apply_rope(queries, cos, sin)
    keys = apply_rope(keys, cos, sin)

    # Expand K and V to match number of heads
    keys = keys.repeat_interleave(self.group_size, dim=1)
    values = values.repeat_interleave(self.group_size, dim=1)

    # Attention
    attn_scores = queries @ keys.transpose(2, 3)
    # ...
```

As mentioned earlier, together with Post-Norm, QK-Norm stabilizes the trai
Note that QK-Norm was not invented by OLMo 2 but goes back to the 202
Scaling Vision Transformers paper.

如前文所述，与后置归一化（Post-Norm）一样，QK归一化（QK-Norm）也能稳
练过程。需要注意的是，QK归一化并非由OLMo 2发明，而是源自《2023年扩展
Transformer论文》。

2.3 OLMo 2 Summary 2.3 OLMo 2总结

In short, the noteworthy OLMo 2 architecture design decisions are primarily
RMSNorm placements: RMSNorm after instead of before the attention and
FeedForward modules (a flavor of Post-Norm), as well as the addition of
RMSNorm for the queries and keys inside the attention mechanism (QK-No
which both, together, help stabilize the training loss.

简而言之，值得关注的OLMo 2架构设计决策主要在于RMSNorm的位置安排：将
RMSNorm置于注意力模块和前馈模块之后而非之前（一种后置归一化方式），以
注意力机制中对查询值和键值添加RMSNorm（QK归一化），这两种方式共同有助
稳定训练损失。

Below is a figure that further compares OLMo 2 to Llama 3 side by side; as
can see, the architectures are otherwise relatively similar except for the fact

OLMo 2 still uses the traditional MHA instead of GQA. (However, the OLMo team released a 32B variant 3 months later that uses GQA.)

以下是一张进一步将OLMo 2与Llama 3进行并排比较的图；可以看到，除了OLMo仍使用传统的多头注意力机制（MHA）而非分组查询注意力机制（GQA）之外，这种架构在其他方面相对相似。（不过，OLMo 2团队在三个月后发布了一个使用GQA的320亿参数版本。）

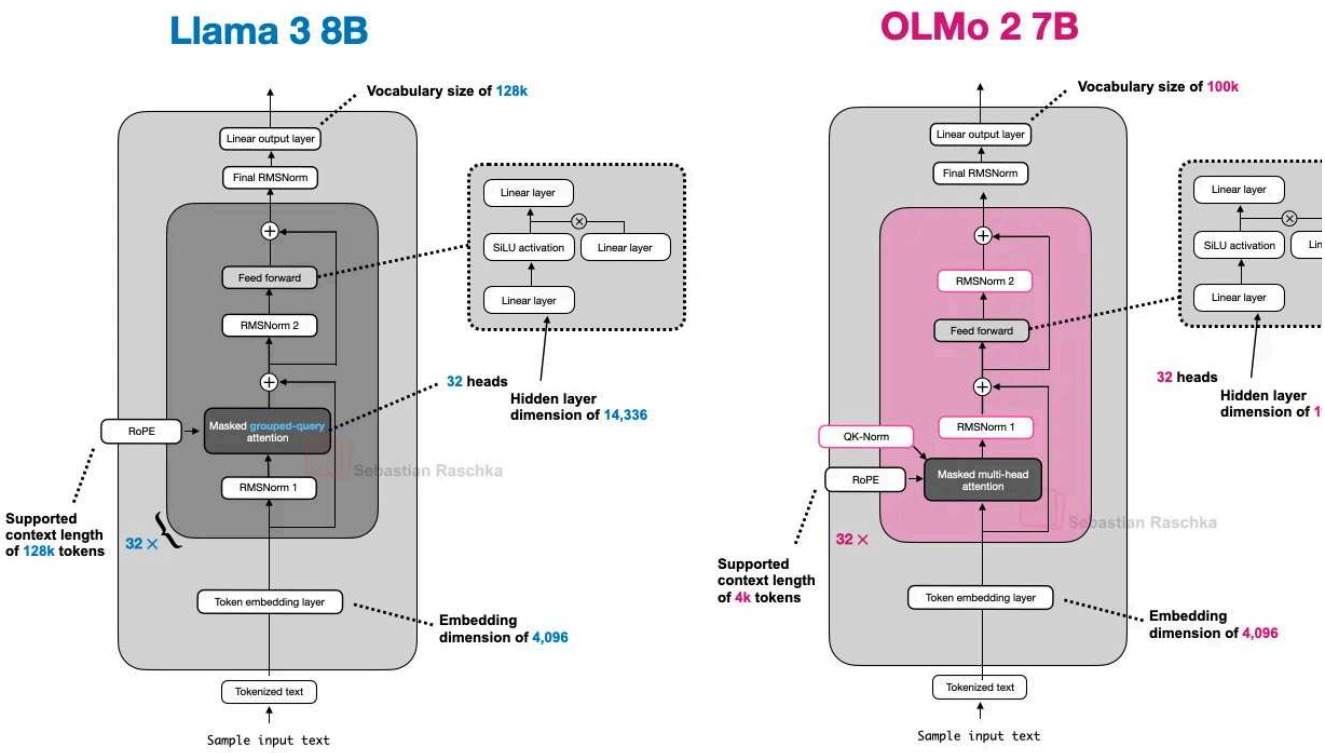


Figure 10: An architecture comparison between Llama 3 and OLMo 2.

图10：Llama 3与OLMo 2的架构对比。

3. Gemma 3 3.宝石3

Google's Gemma models have always been really good, and I think they have always been a bit underhyped compared to other popular models, like the Llama series.

谷歌的Gemma模型一直都非常出色，我认为与其他热门模型（如Llama系列）相比，它们的宣传力度一直略显不足。

One of the distinguishing aspects of Gemma is the rather large vocabulary s (to support multiple languages better), and the stronger focus on the 27B s (versus 8B or 70B). But note that Gemma 2 also comes in smaller sizes: 1B, 4 and 12B.

杰玛（Gemma）的显著特点之一是词汇量相当大（以便更好地支持多种语言），并 更侧重于270亿参数规模（相对于80亿或700亿参数规模）。但请注意，杰玛2（Gemma 2）也有较小的参数规模版本：10亿、40亿和120亿。

The 27B size hits a really nice sweet spot: it's much more capable than an 8B model but not as resource-intensive as a 70B model, and it runs just fine loc on my Mac Mini.

270亿参数规模达到了一个非常理想的平衡点：它的能力比80亿参数的模型强得多，又不像700亿参数的模型那样耗费资源，而且在我的Mac Mini上本地运行也完全没题。

So, what else is interesting in Gemma 3? As discussed earlier, other models Deepseek-V3/R1 use a Mixture-of-Experts (MoE) architecture to reduce mer requirements at inference, given a fixed model size. (The MoE approach is a used by several other models we will discuss later.)

那么，Gemma 3还有哪些有趣的地方呢？如前文所述，像Deepseek-V3/R1等其他型，在模型规模固定的情况下，采用混合专家（MoE）架构来降低推理时的内存需（我们稍后会讨论的其他几个模型也采用了MoE方法。）

Gemma 3 uses a different "trick" to reduce computational costs, namely slic window attention.

Gemma 3采用了一种不同的“技巧”来降低计算成本，即滑动窗口注意力机制。

3.1 Sliding Window Attention
3.1 滑动窗口注意力机制

With sliding window attention (originally introduced in the LongFormer pap in 2020 and also already used by Gemma 2), the Gemma 3 team was able to reduce the memory requirements in the KV cache by a substantial amount,

shown in the figure below.

通过滑动窗口注意力机制（最初于2020年在《LongFormer论文》中提出，并且已
Gemma 2中使用），Gemma 3团队能够大幅降低KV缓存中的内存需求，如下图所
示。

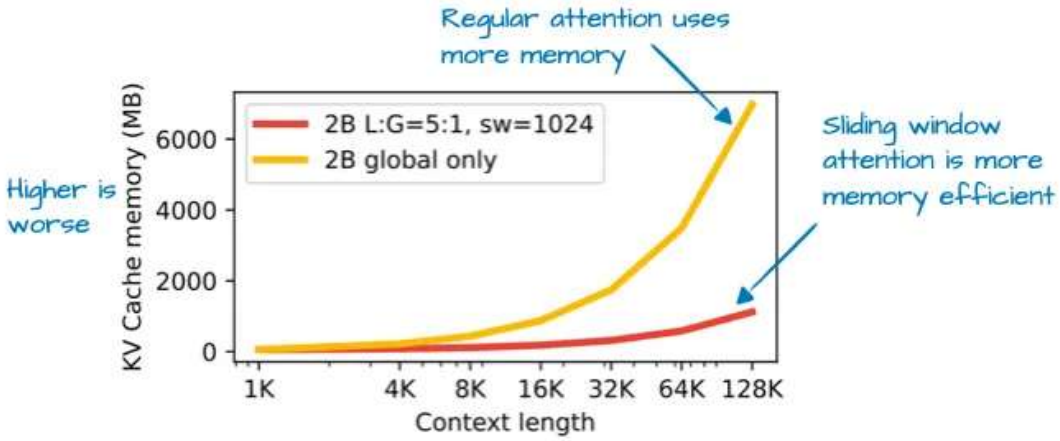


Figure 11: An annotated figure from Gemma 3 paper (https://arxiv.org/abs/2503.19786) showing the KV cache memory savings via sliding window attention.

图11：来自Gemma 3论文（https://arxiv.org/abs/2503.19786）的注释图，展示了通过滑动窗口注意力实现的键值缓存内存节省。

So, what is sliding window attention? If we think of regular self-attention as *global* attention mechanism, since each sequence element can access every other sequence element, then we can think of sliding window attention as *l* attention, because here we restrict the context size around the current quer position. This is illustrated in the figure below.

那么，什么是滑动窗口注意力机制呢？如果我们将常规的自注意力机制视为一种全
意力机制，因为每个序列元素都可以访问其他所有序列元素，那么我们可以将滑动
注意力机制视为*局部*注意力机制，因为在此我们限制了当前查询位置周围的上下文
小。如下图所示。

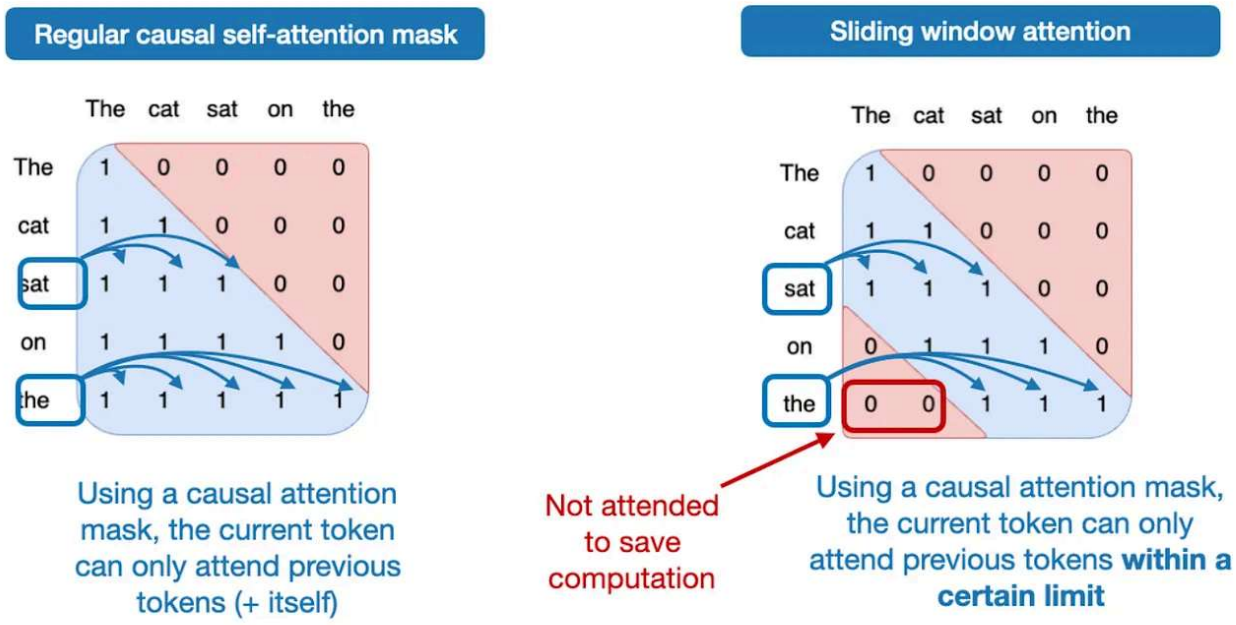


Figure 12: A comparison between regular attention (left) and sliding window attention (right).

图12：常规注意力（左）与滑动窗口注意力（右）的对比。

Please note that sliding window attention can be used with both Multi-Hea Attention and Grouped-Query Attention; Gemma 3 uses grouped-query attention.

请注意，滑动窗口注意力可以与多头注意力和分组查询注意力一起使用；Gemma 用分组查询注意力。

As mentioned above, sliding window attention is also referred to as *local* attention because the local window surrounds and moves with the current query position. In contrast, regular attention is *global* as each token can acc all other tokens.

如上文所述，滑动窗口注意力也被称为*局部*注意力，因为局部窗口围绕当前查询位 随其移动。相比之下，常规注意力是*全局*的，因为每个词元都可以访问所有其他词

Now, as briefly mentioned above, the Gemma 2 predecessor architecture al used sliding window attention before. The difference in Gemma 3 is that the adjusted the ratio between global (regular) and local (sliding) attention.

如上文简要提及，前代架构Gemma 2之前也采用了滑动窗口注意力机制。Gemma
不同之处在于，它调整了全局（常规）注意力与局部（滑动）注意力之间的比例。

For instance, Gemma 2 uses a hybrid attention mechanism that combines sliding window (local) and global attention in a 1:1 ratio. Each token can att to a 4k-token window of nearby context.

例如，Gemma 2使用一种混合注意力机制，该机制以1:1的比例结合了滑动窗口（
部）注意力和全局注意力。每个词元都可以关注到附近4000个词元的上下文窗口。

Where Gemma 2 used sliding window attention in every other layer, Gemma now has a 5:1 ratio, meaning there's only 1 full attention layer for every 5 sliding windows (local) attention layers; moreover, the sliding window size w reduced from 4096 (Gemma 2) to just 1024 (Gemma 3). This shifts the mode focus towards more efficient, localized computations.

在Gemma 2中，每隔一层使用滑动窗口注意力机制，而Gemma 3现在采用了5:1的
例，这意味着每5个滑动窗口（局部）注意力层对应只有1个全注意力层；此外，滑
口大小从4096（Gemma 2）减小到了仅1024（Gemma 3）。这使得模型的关注
向更高效的局部计算。

According to their ablation study, the use of sliding window attention has minimal impact on modeling performance, as shown in the figure below.

根据他们的对比研究，如图所示，使用滑动窗口注意力对建模性能的影响微乎其微

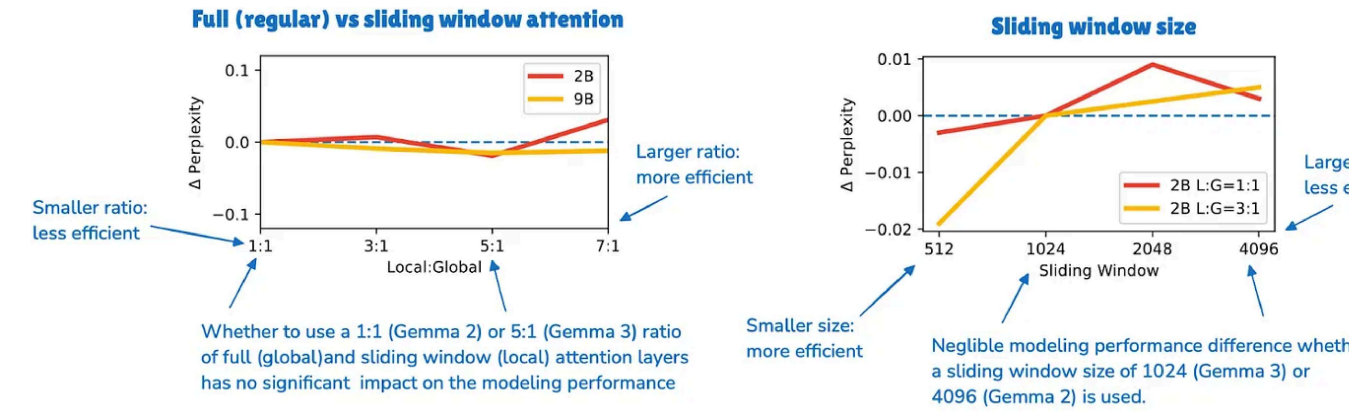


Figure 13: An annotated figure from Gemma 3 paper (https://arxiv.org/abs/2503.19786) showing that sliding window attention has

little to no impact on the LLM-generated output perplexity.

图13：来自Gemma 3论文（https://arxiv.org/abs/2503.19786）的注释图，表明滑动窗口注意力对大语言模型生成的输出困惑度几乎没有影响。

While sliding window attention is the most notable architecture aspect of Gemma 3, I want to also briefly go over the placement of the normalization layers as a follow-up to the previous OLMo 2 section.

虽然滑动窗口注意力机制是Gemma 3最显著的架构特点，但作为对上一章节OLM的补充，我也想简要介绍一下归一化层的布局。

3.2 Normalization Layer Placement in Gemm 3

3.2 杰玛3（Gemma 3）中归一化层的位置

A small but interesting tidbit to highlight is that Gemma 3 uses RMSNorm i both a Pre-Norm and Post-Norm setting around its grouped-query attentio module.

有一个虽小但有趣的细节值得强调：Gemma 3在其分组查询注意力模块周围的前置一化和后置归一化设置中都使用了均方根归一化（RMSNorm）。

This is similar to Gemma 2 but still worth highlighting, as it differs from (1) t Post-Norm used in the original transformer (“Attention is all you need”), (2) Pre-Norm, which was popularized by GPT-2 and used in many other architectures afterwards, and (3) the Post-Norm flavor in OLMo 2 that we sa earlier.

这与Gemma 2类似，但仍值得强调，因为它不同于（1）原始Transformer（《Attention is all you need》）中使用的后置归一化（Post-Norm），（2）由GPT-2推广并在随后的许多其他架构中使用的前置归一化（Pre-Norm），以及（3）我们之前看到的OLMo 2中的后置归一化（Post-Norm）风格。

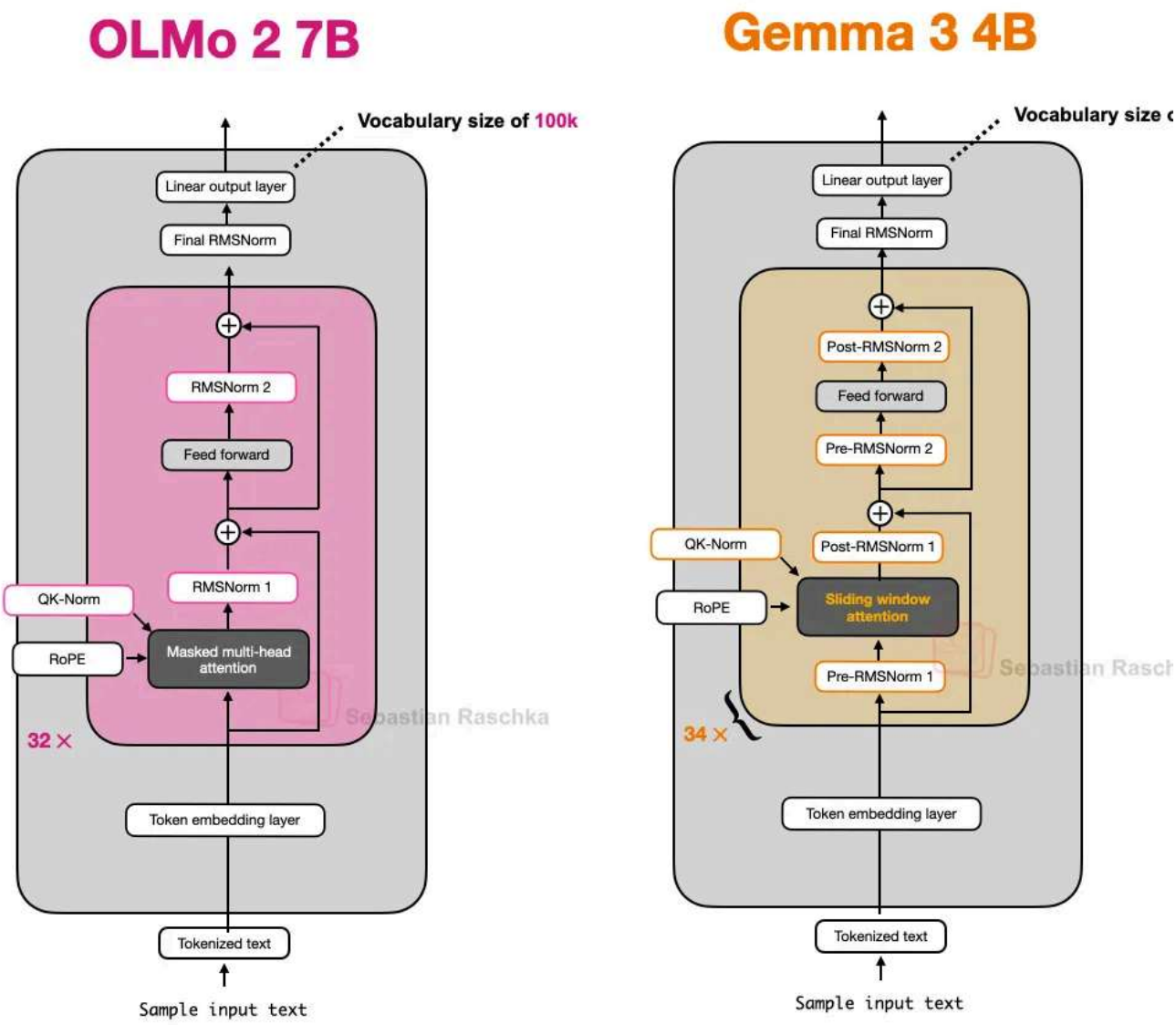


Figure 14: An architecture comparison between OLMo2 and Gemma 3; note the additional normalization layers in Gemma 3.

图14：OLMo2与Gemma 3的架构对比；请注意Gemma 3中额外的归一化层。

I think this normalization layer placement is a relatively intuitive approach a gets the best of both worlds: Pre-Norm and Post-Norm. In my opinion, a bit extra normalization can't hurt. In the worst case, if the extra normalization is redundant, this adds a bit of inefficiency through redundancy. In practice, si RMSNorm is relatively cheap in the grand scheme of things, this shouldn't h any noticeable impact, though.

我认为这种归一化层的放置方式是一种相对直观的方法，因为它兼顾了两种方式的点：前置归一化和后置归一化。在我看来，多一点额外的归一化并无坏处。在最坏况下，如果额外的归一化是多余的，这会因冗余而增加一点低效性。不过在实际应

中，由于均方根归一化（RMSNorm）从整体来看成本相对较低，所以这不应该有
明显的影响。

3.3 Gemma 3 Summary 3.3 Gemma 3总结

Gemma 3 is a well-performing open-weight LLM that, in my opinion, is a bi
underappreciated in the open-source circles. The most interesting part is th
use of sliding window attention to improve efficiency (it will be interesting t
combine it with MoE in the future).

杰玛3是一款性能出色的开源大语言模型，在我看来，它在开源领域有点被低估了。
有意思的部分是它使用滑动窗口注意力机制来提高效率（未来将其与专家混合模型
合会很有趣）。

Also, Gemma 3 has a unique normalization layer placement, placing RMSNo
layers both before and after the attention and FeedForward modules.

此外，Gemma 3具有独特的归一化层设置，在注意力模块和前馈模块之前和之后都
置了RMSNorm层。

3.4 Bonus: Gemma 3n 3.4奖金：珍玛3n

A few months after the Gemma 3 release, Google shared Gemma 3n, which
Gemma 3 model that has been optimized for small-device efficiency with th
goal of running on phones.

杰玛3发布几个月后，谷歌推出了杰玛3n，这是一款针对小型设备效率进行优化的
3模型，目标是在手机上运行。

One of the changes in Gemma 3n to achieve better efficiency is the so-calle
Per-Layer Embedding (PLE) parameters layer. The key idea here is to keep or
subset of the model's parameters in GPU memory. Token-layer specific
embeddings, such as those for text, audio, and vision modalities, are then
streamed from the CPU or SSD on demand.

为了在Gemma 3n中实现更高的效率，其中一项改变是引入了所谓的每层嵌入（PI
参数层。其核心思路是仅在GPU内存中保留模型参数的一个子集。然后，针对特定

token层的嵌入，比如文本、音频和视觉模态的嵌入，会根据需求从CPU或固态硬盘
实时传输。

The figure below illustrates the PLE memory savings, listing 5.44 billion parameters for a standard Gemma 3 model. This likely refers to the Gemma billion variant.

下图展示了PLE在节省内存方面的效果，列出了标准Gemma 3模型的54.4亿个参数
可能指的是40亿参数版本的Gemma 3。

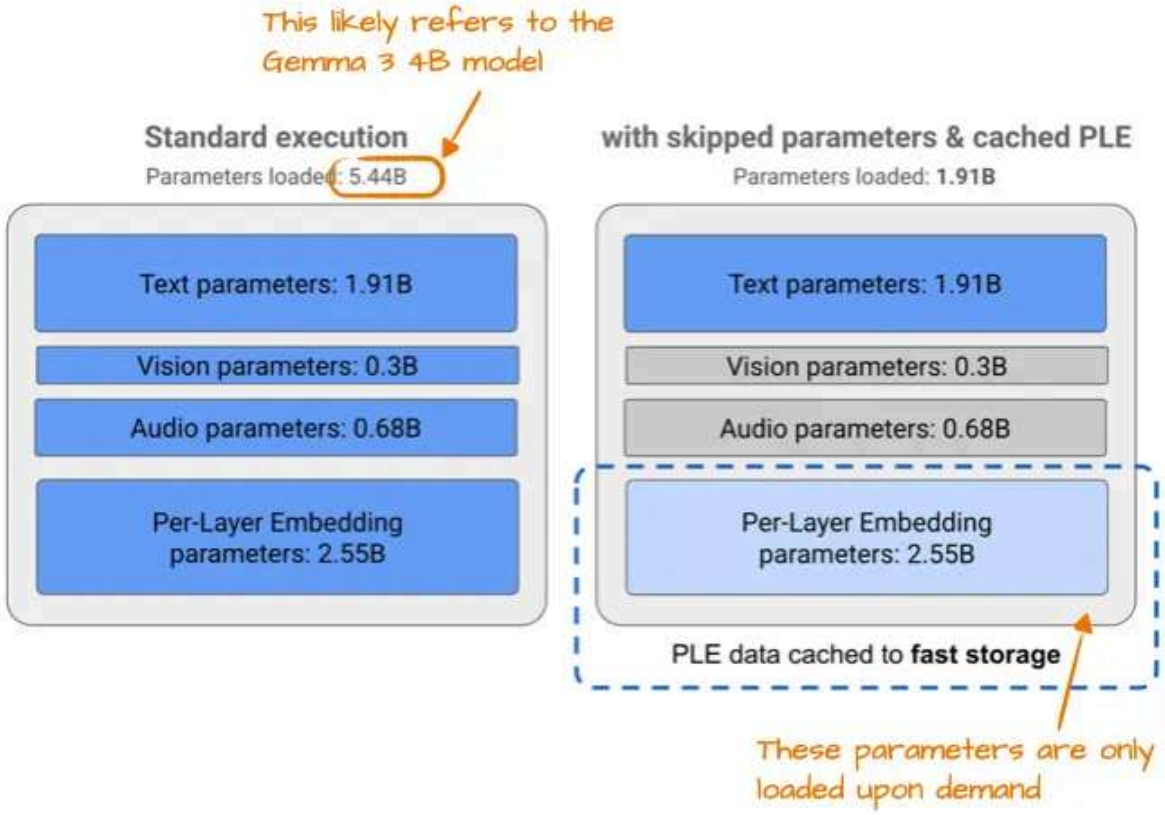


Figure 15: An annotated figure from Google's Gemma 3n blog (https://developers.googleblog.com/en/introducing-gemma-3n/) illustrating the PLE memory savings.

图15：来自谷歌Gemma 3n博客（https://developers.googleblog.com/en/introducing-gemma-3n/）的一张注释图，展示了PLE内存节省情况。

The 5.44 vs. 4 billion parameter discrepancy is because Google has an interesting way of reporting parameter counts in LLMs. They often exclude embedding parameters to make the model appear smaller, except in cases l this, where it is convenient to include them to make the model appear large

This is not unique to Google, as this approach has become a common pract across the field.

54.4亿与40亿参数之间的差异，是因为谷歌在报告大语言模型（LLMs）的参数数量 采用了一种有趣的方式。他们常常会排除嵌入参数，以使模型看起来规模更小，但 种情况下除外，此时将嵌入参数计算在内会让模型看起来规模更大。这种做法并非 独有，因为它已成为整个领域的普遍做法。

Another interesting trick is the MatFormer concept (short for Matryoshka Transformer). For instance, Gemma 3n uses a single shared LLM (transforme architecture that can be sliced into smaller, independently usable models. Ea slice is trained to function on its own, so at inference time, we can run just t part you need (instead of the large model).

另一个有趣的技巧是MatFormer概念（Matryoshka Transformer的缩写）。例如 Gemma 3n使用单一共享的大语言模型（Transformer）架构，该架构可以分割成 的、可独立使用的模型。每个切片都经过单独训练，因此在推理时，我们可以只运 需的部分（而不是整个大模型）。

4. Mistral Small 3.1
4. 米斯特拉尔小型模型3.1

Mistral Small 3.1 24B, which was released in March shortly after Gemma 3, i noteworthy for outperforming Gemma 3 27B on several benchmarks (excep math) while being faster.

Mistral Small 3.1 240亿参数版本于3月发布，紧随Gemma 3之后。值得注意的是 模型在多个基准测试（数学除外）中表现优于Gemma 3 270亿参数版本，且速度更 快。

The reasons for the lower inference latency of Mistral Small 3.1 over Gemma are likely due to their custom tokenizer, as well as shrinking the KV cache ar layer count. Otherwise, it's a standard architecture as shown in the figure be

Mistral Small 3.1推理延迟低于Gemma 3，原因可能在于其定制的分词器，以及缩的键值（KV）缓存和层数。除此之外，它采用的是如下图所示的标准架构。

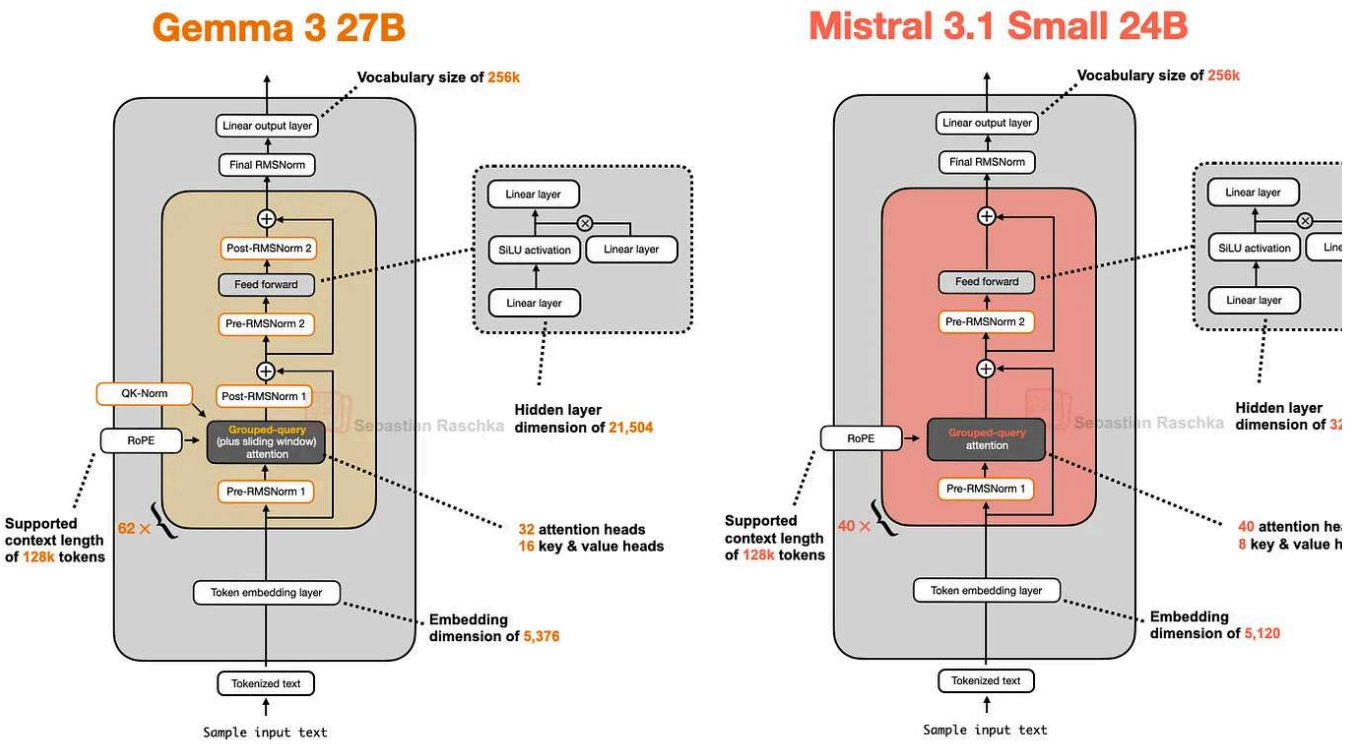


Figure 16: An architecture comparison between Gemma 3 27B and Mistral 3.1 Small 24B.

图16：Gemma 3 270亿参数模型与Mistral 3.1 Small 240亿参数模型的架构对比。

Interestingly, earlier Mistral models had utilized sliding window attention, b they appear to have abandoned it in Mistral Small 3.1. So, since Mistral uses regular Grouped-Query Attention instead of Grouped-Query Attention with sliding window as in Gemma 3, maybe there are additional inference compu savings due to being able to use more optimized code (i.e., FlashAttention). instance, I speculate that while sliding window attention reduces memory usage, it doesn't necessarily reduce inference latency, which is what Mistral Small 3.1 is focused on.

有趣的是，早期的Mistral模型曾使用滑动窗口注意力机制，但在Mistral Small 3.1 乎已将其舍弃。因此，由于Mistral使用常规的分组查询注意力机制，而非像Gemm 那样采用带滑动窗口的分组查询注意力机制，或许因能够使用更优化的代码（如 FlashAttention）而在推理计算方面进一步节省资源。例如，我推测虽然滑动窗口

力机制可减少内存使用量，但未必能降低推理延迟，而这正是Mistral Small 3.1所
的重点。

5. Llama 4 5. Llama 4（小羊驼4模型）

The extensive introductory discussion on Mixture-of-Experts (MoE) earlier ir this article pays off again. Llama 4 has also adopted an MoE approach and otherwise follows a relatively standard architecture that is very similar to DeepSeek-V3, as shown in the figure below. (Llama 4 includes native multimodal support, similar to models like Gemma and Mistral. However, si this article focuses on language modeling, we only focus on the text model.

本文前面关于专家混合（MoE）的广泛介绍性讨论再次得到了回报。 Llama 4 也采了MoE方法，除此之外，它遵循一种相对标准的架构，与DeepSeek-V3非常相似，下图所示。（Llama 4包括原生多模态支持，类似于Gemma和Mistral等模型。然而由于本文专注于语言建模，我们只关注文本模型。）

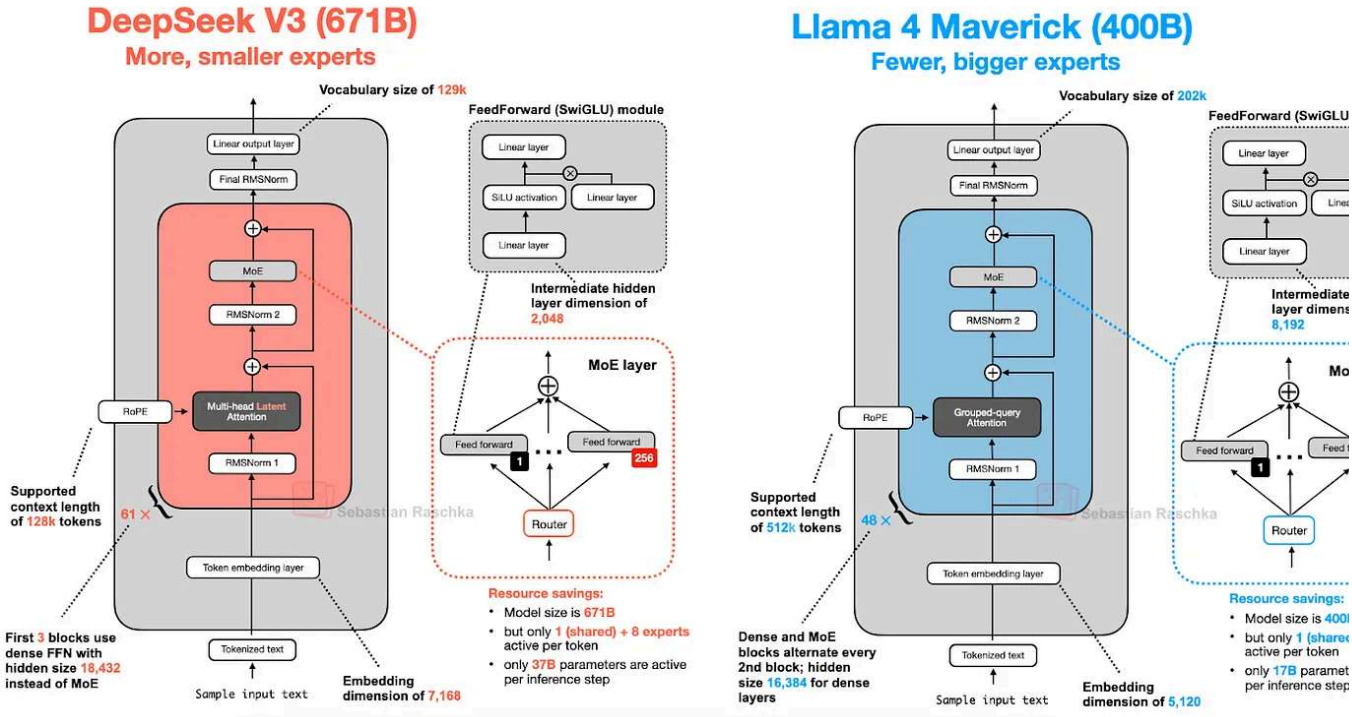


Figure 17: An architecture comparison between DeepSeek V3 (671-billion parameters) and Llama 4 Maverick (400-billion parameters).

图17：DeepSeek V3（6710亿参数）与Llama 4 Maverick（4000亿参数）的架构对比。

While the Llama 4 Maverick architecture looks very similar to DeepSeek-V3 overall, there are some interesting differences worth highlighting.

虽然Llama 4 Maverick架构总体上看起来与DeepSeek-V3非常相似，但仍有一些值得强调的有趣差异。

First, Llama 4 uses Grouped-Query Attention similar to its predecessors, whereas DeepSeek-V3 uses Multi-Head Latent Attention, which we discusse the beginning of this article. Now, both DeepSeek-V3 and Llama 4 Maverick very large architectures, with DeepSeek-V3 being approximately 68% larger its total parameter count. However, with 37 billion active parameters, DeepS V3 has more than twice as many active parameters as Llama 4 Maverick (17

首先，Llama 4与其前身类似，采用分组查询注意力机制，而DeepSeek-V3则采用潜在注意力机制，我们在本文开头讨论过这一点。如今，DeepSeek-V3和Llama 4 Maverick都是非常庞大的架构，DeepSeek-V3的总参数数量大约多68%。然而，DeepSeek-V3拥有370亿个活跃参数，其活跃参数数量是Llama 4 Maverick（170的两倍多。

Llama 4 Maverick uses a more classic MoE setup with fewer but larger expe (2 active experts with 8,192 hidden size each) compared to DeepSeek-V3 (9 active experts with 2,048 hidden size each). Also, DeepSeek uses MoE layers each transformer block (except the first 3), whereas Llama 4 alternates MoE dense modules in every other transformer block.

Llama 4 Maverick采用了更经典的混合专家（MoE）设置，与DeepSeek-V3相比，专家数量更少但规模更大（2个激活专家，每个专家的隐藏层大小为8192），而DeepSeek-V3有9个激活专家，每个专家的隐藏层大小为2048。此外，DeepSeek个Transformer模块（除前3个外）中都使用了MoE层，而Llama 4则在每隔一个Transformer模块中交替使用MoE和密集模块。

Given the many small differences between architectures, it is difficult to determine their exact impact on final model performance. The main takeaw however, is that MoE architectures have seen a significant rise in popularity

2025.

鉴于不同架构之间存在诸多细微差异，很难确定它们对最终模型性能的确切影响。
而，主要结论是，混合专家（MoE）架构在2025年的受欢迎程度显著上升。

6. Qwen3

The Qwen team consistently delivers high-quality open-weight LLMs. When helped co-advising the LLM efficiency challenge at NeurIPS 2023, I rememb that the top winning solutions were all Qwen2-based.

通义千问团队始终提供高质量的开源大语言模型。当我在2023年神经信息处理系统
会（NeurIPS 2023）上协助共同指导大语言模型效率挑战赛时，我记得名列前茅的
奖方案均基于通义千问2。

Now, Qwen3 is another hit model series at the top of the leaderboards for t size classes. There are 7 dense models: 0.6B, 1.7B, 4B, 8B, 14B, and 32B. And there are 2 MoE models: 30B-A3B, and 235B-A22B.

如今，通义千问3（Qwen3）是另一款热门模型系列，在其规模类别排行榜上名列
茅。共有7个稠密模型：6亿、17亿、40亿、80亿、140亿和320亿参数。还有2个
专家（MoE）模型：300亿-A30亿，以及2350亿-A220亿。

(By the way, note that the missing whitespace in "Qwen3" is not a typo; I sin try to preserve the original spelling the Qwen developers chose.)

（顺便说一下，请注意"Qwen3"中缺少空格并非拼写错误；我只是尽量保留Qwen
者选择的原始拼写。）

6.1 Qwen3 (Dense) 6.1 通义千问3（密集型）

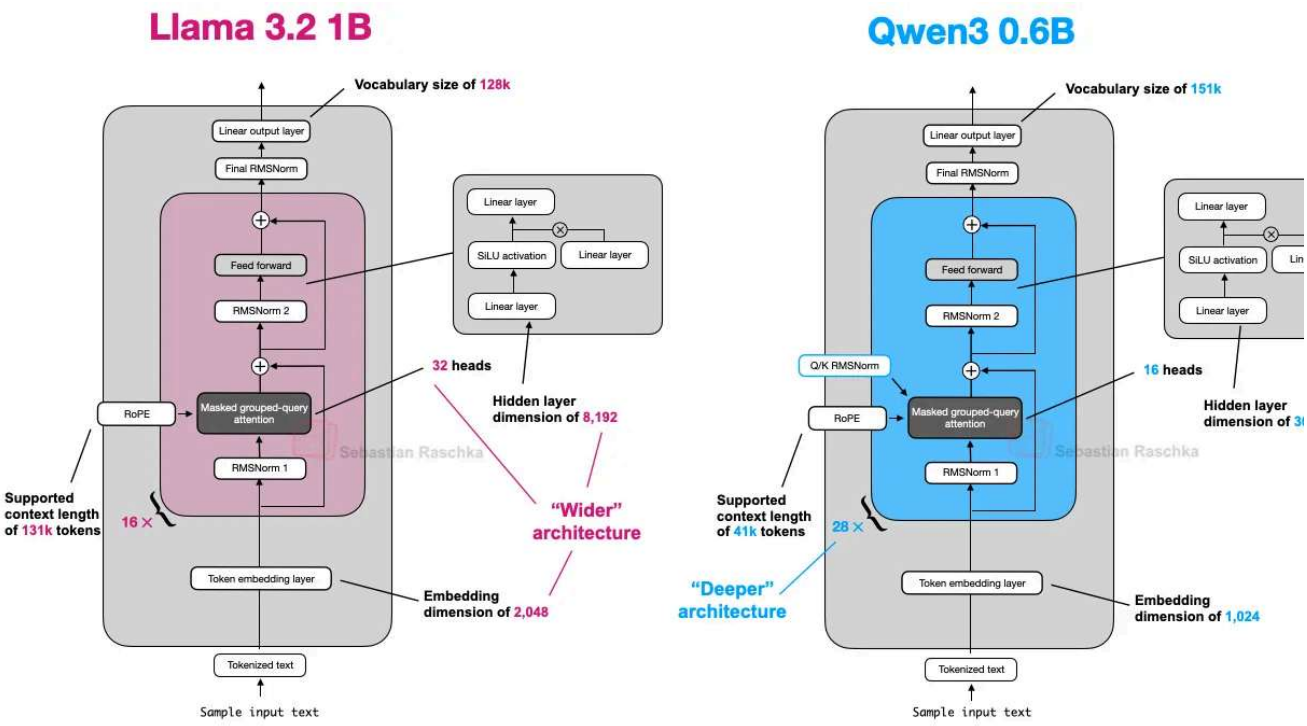
Let's discuss the dense model architecture first. As of this writing, the 0.6B model may well be the smallest current-generation open-weight model out there. And based on my personal experience, it performs really well given it small size. It has great token/sec throughput and a low memory footprint if are planning to run it locally. But what's more, it's also easy to train locally (f

educational purposes) due to its small size.

我们先来讨论密集模型架构。在撰写本文时，6亿参数的模型很可能是目前最小的当开源权重模型。根据我个人的经验，考虑到其规模较小，它的表现相当出色。如果算在本地运行它，它具有很高的每秒令牌处理量和较低的内存占用。但更重要的是于其规模较小，在本地训练（出于教学目的）也很容易。

So, Qwen3 0.6B has replaced Llama 3 1B for me for most purposes. A comparison between these two architectures is shown below.

所以，在大多数情况下，Qwen3 0.6B已经取代Llama 3 1B成为我的首选。以下是种架构的对比。



	Tokens/sec	Memory
Llama3Model 1B	42	2.91 GB
Llama3Model 1B compiled	170	3.12 GB

	Tokens/sec	Memory
Qwen3Model 0.6B	24	1.49 GB
Qwen3Model 0.6B compiled	101	1.99 GB

Figure 18: An architecture comparison between Qwen3 0.6B and Llama 3 1B; notice that Qwen3 is a deeper architecture with more layers, whereas Llama 3 is a wider architecture with more attention heads.

图18：Qwen3 0.6B与Llama 3 10亿参数模型的架构对比；请注意，Qwen3是一种更深层次的架构，具有更多的层，而Llama 3是一种更宽的架构，具有更多的注意力头。

If you are interested in a human-readable Qwen3 implementation without external third-party LLM library dependencies, I recently implemented Qwe

from scratch (in pure PyTorch).

如果你对不依赖外部第三方大语言模型库且便于人类阅读的Qwen3实现感兴趣，我近从零开始（纯PyTorch实现）实现了Qwen3。

The computational performance numbers in the figure above are based on from-scratch PyTorch implementations when run on an A100 GPU. As one c see, Qwen3 has a smaller memory footprint as it is a smaller architecture overall, but also uses smaller hidden layers and fewer attention heads. How it uses more transformer blocks than Llama 3, which leads to a slower runtir (lower tokens/sec generation speed).

上图中的计算性能数据基于我在A100 GPU上运行的从头开始编写的PyTorch实现。如人们所见，Qwen3的内存占用较小，因为它整体架构较小，同时使用的隐藏层和意力头也较少。然而，它使用的Transformer模块比Llama 3更多，这导致运行时间慢（每秒生成的令牌数较低）。

6.2 Qwen3 (MoE) 6.2 通义千问3（混合专家模型

As mentioned earlier, Qwen3 also comes in two MoE flavors: 30B-A3B and 235B-A22B. Why do some architectures, like Qwen3, come as regular (dense and MoE (sparse) variants?

如前所述，通义千问3也有两种混合专家（MoE）版本：300亿参数-30亿专家参数（30B-A3B）和2350亿参数-220亿专家参数（235B-A22B）。为什么有些架构，通义千问3，会有常规（密集）和混合专家（稀疏）两种变体呢?

As mentioned at the beginning of this article, MoE variants help reduce inference costs for large base models. Offering both dense and MoE versior gives users flexibility depending on their goals and constraints.

如本文开头所述，混合专家（MoE）变体有助于降低大型基础模型的推理成本。同供密集型和混合专家（MoE）版本，使用户可以根据自身目标和限制灵活选择。

Dense models are typically more straightforward to fine-tune, deploy, and optimize across various hardware.

稠密模型通常更容易在各种硬件上进行微调、部署和优化。

On the other hand, MoE models are optimized for scaling inference. For instance, at a fixed inference budget, they can achieve a higher overall mod capacity (i.e., knowledge uptake during training due to being larger) withou proportionally increasing inference costs.

另一方面，专家混合（MoE）模型针对推理扩展进行了优化。例如，在固定的推理下，它们可以实现更高的整体模型容量（即在训练过程中由于模型规模更大而吸收知识），而不会按比例增加推理成本。

By releasing both types, the Qwen3 series can support a broader range of u cases: dense models for robustness, simplicity, and fine-tuning, and MoE models for efficient serving at scale.

通过发布这两种类型，通义千问3系列能够支持更广泛的用例：稠密模型用于实现稳性、简易性和微调，混合专家（MoE）模型则用于高效的大规模服务。

To round up this section, let's look at Qwen3 235B-A22B (note that the A22I stands for "22B active parameters) to DeepSeek-V3, which has almost twice many active parameters (37B).

在本节结尾，我们来看一下从通义千问3 2350亿参数-220亿激活参数模型（注意，A22B代表“220亿激活参数”）到渊知模型V3，后者的激活参数几乎是前者的两倍（亿）。

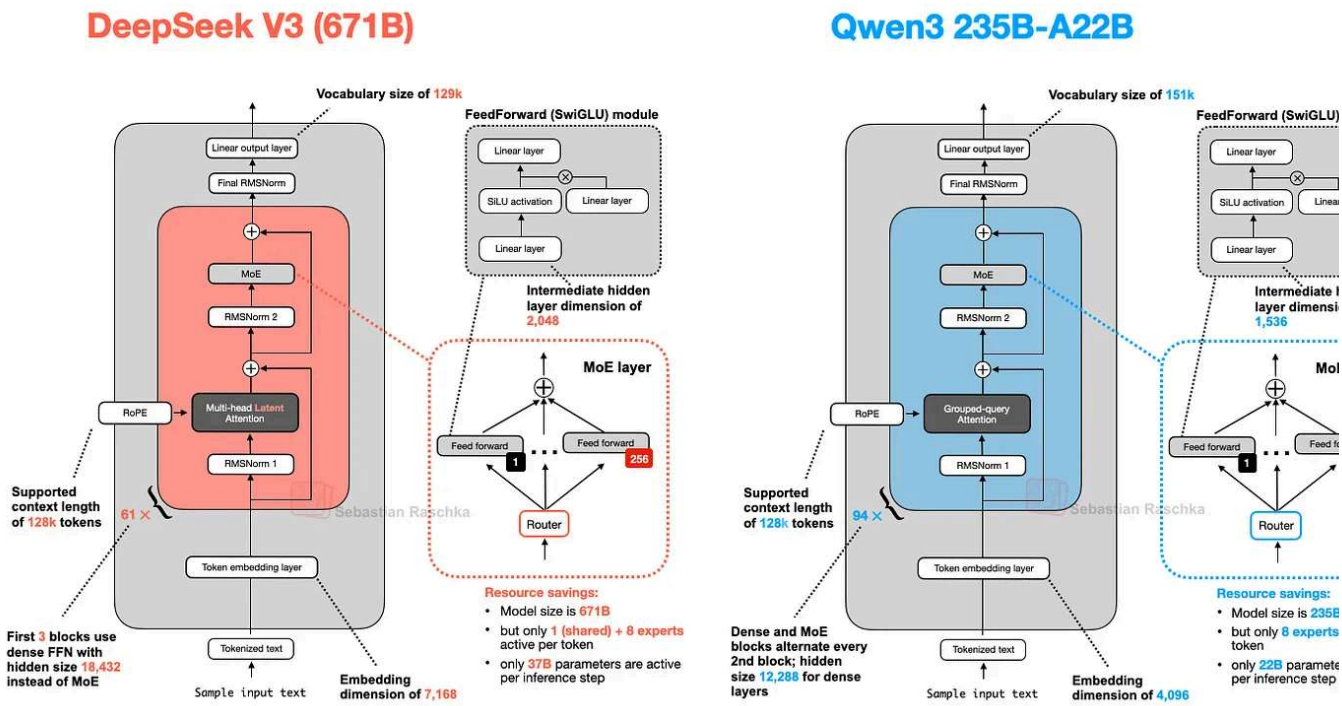


Figure 19: An architecture comparison between DeepSeek-V3 and Qwen3 235B-A22B.

图19：DeepSeek-V3与Qwen3 235B-A22B的架构对比。

As shown in the figure above, the DeepSeek-V3 and Qwen3 235B-A22B architectures are remarkably similar. What's noteworthy, though, is that the Qwen3 model moved away from using a shared expert (earlier Qwen mode such as Qwen2.5-MoE did use a shared expert).

如上图所示，DeepSeek-V3和Qwen3 235B-A22B架构极为相似。不过，值得注意是，Qwen3模型不再使用共享专家（早期的Qwen模型，如Qwen2.5-MoE确实使共享专家）。

Unfortunately, the Qwen3 team did not disclose any reason as to why they moved away from shared experts. If I had to guess, it was perhaps simply no necessary for training stability for their setup when they increased the expe from 2 (in Qwen2.5-MoE) to 8 (in Qwen3). And then they were able to save extra compute/memory cost by using only 8 instead of 8+1 experts. (Howe this doesn't explain why DeepSeek-V3 is still keeping their shared expert.)

不幸的是，通义千问3（Qwen3）团队并未透露他们放弃共享专家的任何原因。如要猜测的话，可能是当他们将专家数量从2个（在通义千问2.5混合专家模型（Qwen2.5-MoE）中）增加到8个（在通义千问3中）时，对于他们的设置而言，训练稳定性，共享专家并非必要。这样一来，他们仅使用8个专家而非8 + 1个专家能节省额外的计算/内存成本。（然而，这并不能解释为什么DeepSeek-V3仍保留共享专家。）

7. SmolLM3
7. 小语言模型3（SmolLM3）

SmolLM3 is perhaps not as nearly as popular as the other LLMs covered in t article, but I thought it is still an interesting model to include as it offers rea good modeling performance at a relatively small and convenient 3-billion parameter model size that sits between the 1.7B and 4B Qwen3 model, as

shown in the figure below.

SmolLM3或许不像本文所提及的其他大语言模型那样受欢迎，但我认为它仍是一个得纳入讨论的有趣模型。从下图可见，它的参数量为30亿，介于17亿参数的Qwen型和40亿参数的Qwen3模型之间，规模相对较小且使用便捷，却能提供相当出色的模性能。

Moreover, it also shared a lot of the training details, similar to OLMo, which rare and always appreciated!

此外，它还分享了许多训练细节，与OLMo类似，这很少见，也总是令人赞赏！

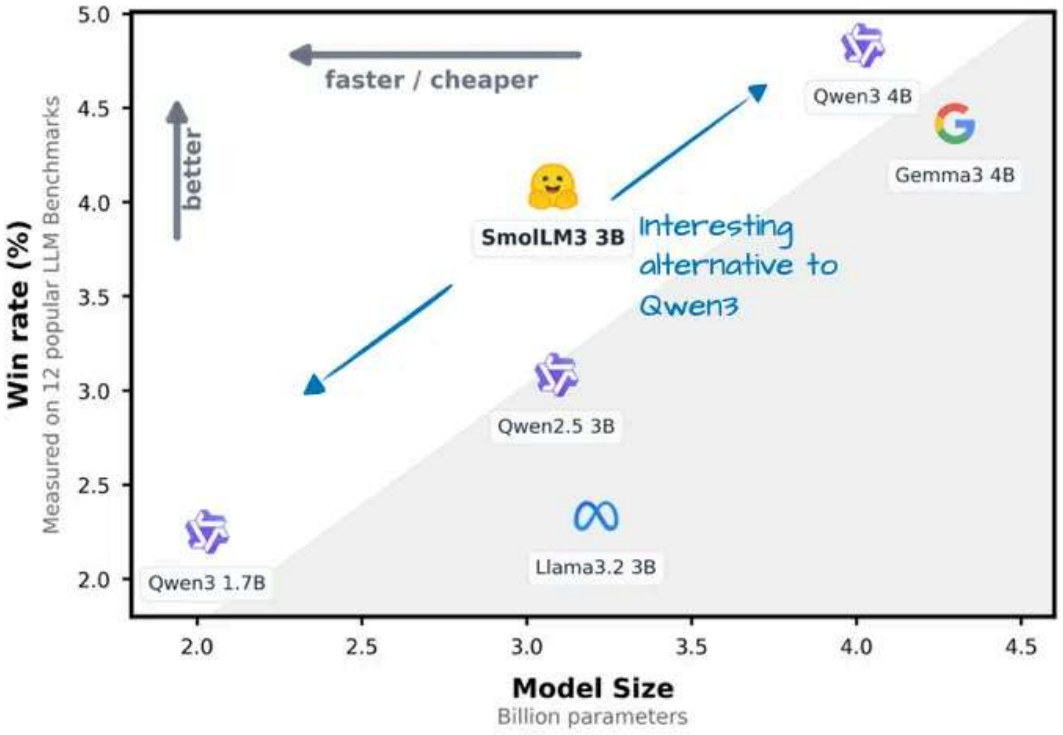


Figure 20: An annotated figure from the SmolLM3 announcement post, https://huggingface.co/blog/smollm3, comparing the SmolLM3 win rate to Qwen3 1.7B and 4B as well as Llama 3 3B and Gemma 3 4B.

图20：来自SmolLM3发布帖子（https://huggingface.co/blog/smollm3）的一张注释图，比较了SmolLM3与Qwen3 17亿和40亿参数模型、Llama 3 30亿参数模型以及Gemma 3 40亿参数模型的胜率。

As shown in the architecture comparison figure below, the SmolLM3 architecture looks fairly standard. The perhaps most interesting aspect is its

of NoPE (No Positional Embeddings), though.

如下方架构对比图所示，SmolLM3架构看起来相当标准。不过，或许最有意思的地
在于它使用了NoPE（无位置嵌入）。

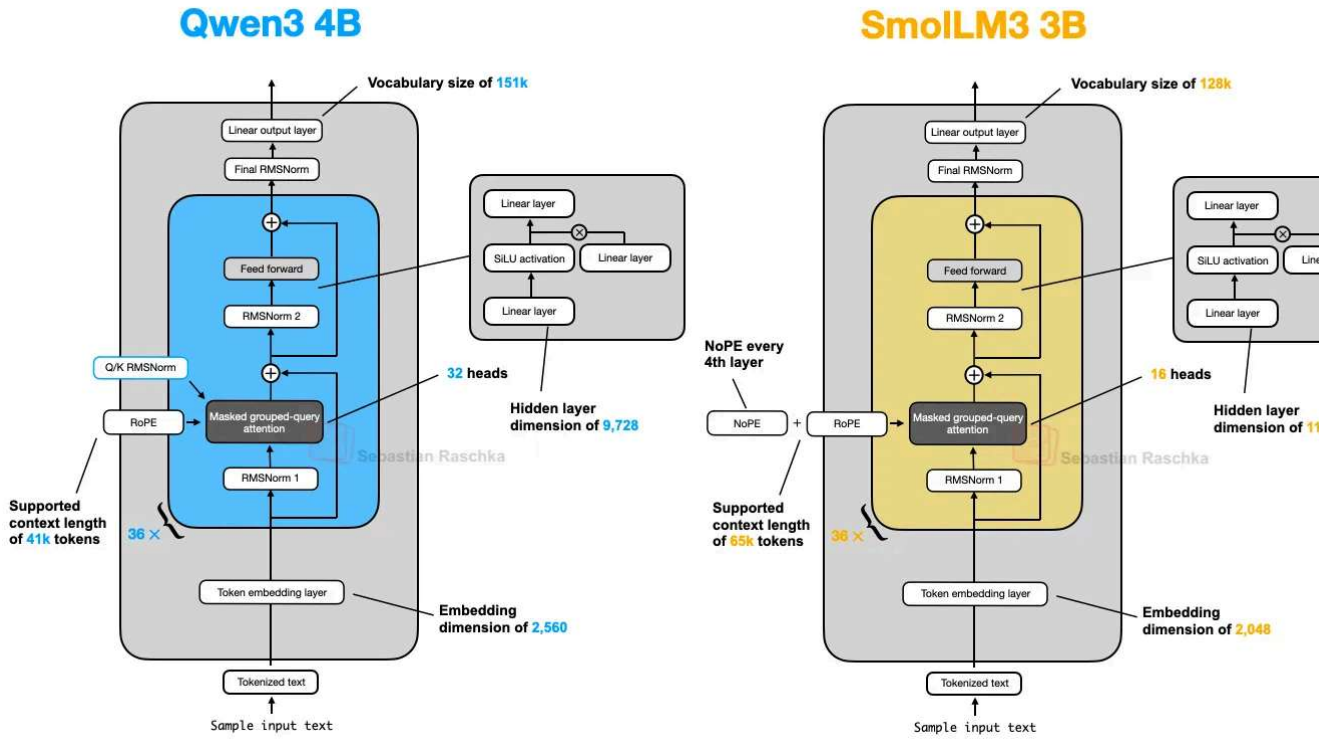


Figure 21: A side-by-side architecture comparison between Qwen3 4B and SmolLM3 3B.

图21：Qwen3 40亿参数模型与SmolLM3 30亿参数模型的架构对比。

7.1 No Positional Embeddings (NoPE)
7.1 无位置嵌入（NoPE）

NoPE is, in LLM contexts, an older idea that goes back to a 2023 paper (The Impact of Positional Encoding on Length Generalization in Transformers) to remove explicit positional information injection (like through classic absolut positional embedding layers in early GPT architectures or nowadays RoPE).

在大语言模型（LLM）语境中，NoPE是一个较早的概念，其起源于2023年的一篇
（《位置编码对Transformer中长度泛化的影响》），该概念旨在去除显式的位置信
注入（例如通过早期GPT架构中经典的绝对位置嵌入层，或如今的旋转位置嵌入
（RoPE））。

In transformer-based LLMs, positional encoding is typically necessary becau self-attention treats tokens independently of order. Absolute position embeddings solve this by adding an additional embedding layer that adds information to the token embeddings.

在基于Transformer的大语言模型（LLMs）中，位置编码通常是必要的，因为自注机制独立于顺序来处理词元。绝对位置嵌入通过添加一个额外的嵌入层来解决这个题，该嵌入层会向词元嵌入添加信息。

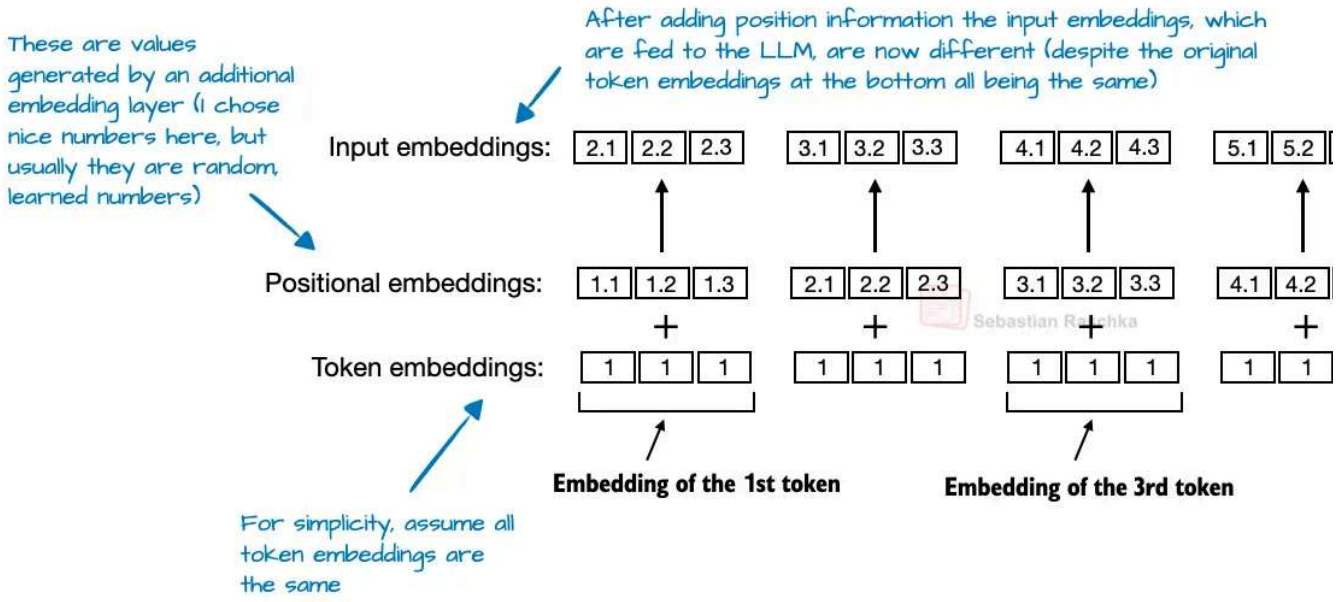


Figure 22: A modified figure from my Build A Large Language Model (From Scratch) book (https://www.amazon.com/Build-Large-Language-Model-Scratch/dp/1633437167) illustrating absolute positional embeddings.

图22：该图改编自我所著的《从零开始构建大语言模型》（https://www.amazon.com/Build-Large-Language-Model-Scratch/dp/1633437167），展示了绝对位置嵌入。

RoPE, on the other hand, solves this by rotating the query and key vectors relative to their token position.

另一方面，旋转位置嵌入（RoPE）通过相对于词元位置旋转查询向量和键向量来解这个问题。

In NoPE layers, however, no such positional signal is added at all: not fixed, learned, not relative. Nothing.

然而，在NoPE层中，根本不会添加任何此类位置信号：既没有固定的，也没有经过
习的，更没有相对的。什么都没有。

Even though there is no positional embedding, the model still knows which
tokens come before, thanks to the causal attention mask. This mask prevent
each token from attending to future ones. As a result, a token at position t c
only see tokens at positions $\leq t$, which preserves the autoregressive orderin

尽管没有位置嵌入，但由于因果注意力掩码的存在，模型仍然知道哪些词元在前。
掩码阻止每个词元关注未来的词元。因此，位置 t 处的词元只能看到位置 $\leq t$ 处的
元，这就保留了自回归顺序。

So while there is no positional information that is explicitly added, there is s
an implicit sense of direction baked into the model's structure, and the LLM
the regular gradient-descent-based training, can learn to exploit it if it finds
beneficial for the optimization objective. (Check out the NoPE paper's theor
for more information.)

因此，虽然没有明确添加位置信息，但模型结构中仍隐含着一种方向感，并且在基
规梯度下降的训练中，大语言模型（LLM）如果发现这对优化目标有益，便能够学
用这种方向感。（欲了解更多信息，请查阅《无位置嵌入（NoPE）》论文中的定
理。）

So, overall, the NoPE paper not only found that no positional information
injection is necessary, but it also found that NoPE has better length
generalization, which means that LLM answering performance deteriorates
with increased sequence length, as shown in the figure below.

因此，总体而言，《无位置嵌入（NoPE）》论文不仅发现无需注入位置信息，还发
NoPE具有更好的长度泛化能力，这意味着大语言模型（LLM）的回答性能随序列长
增加而下降的幅度更小，如下图所示。

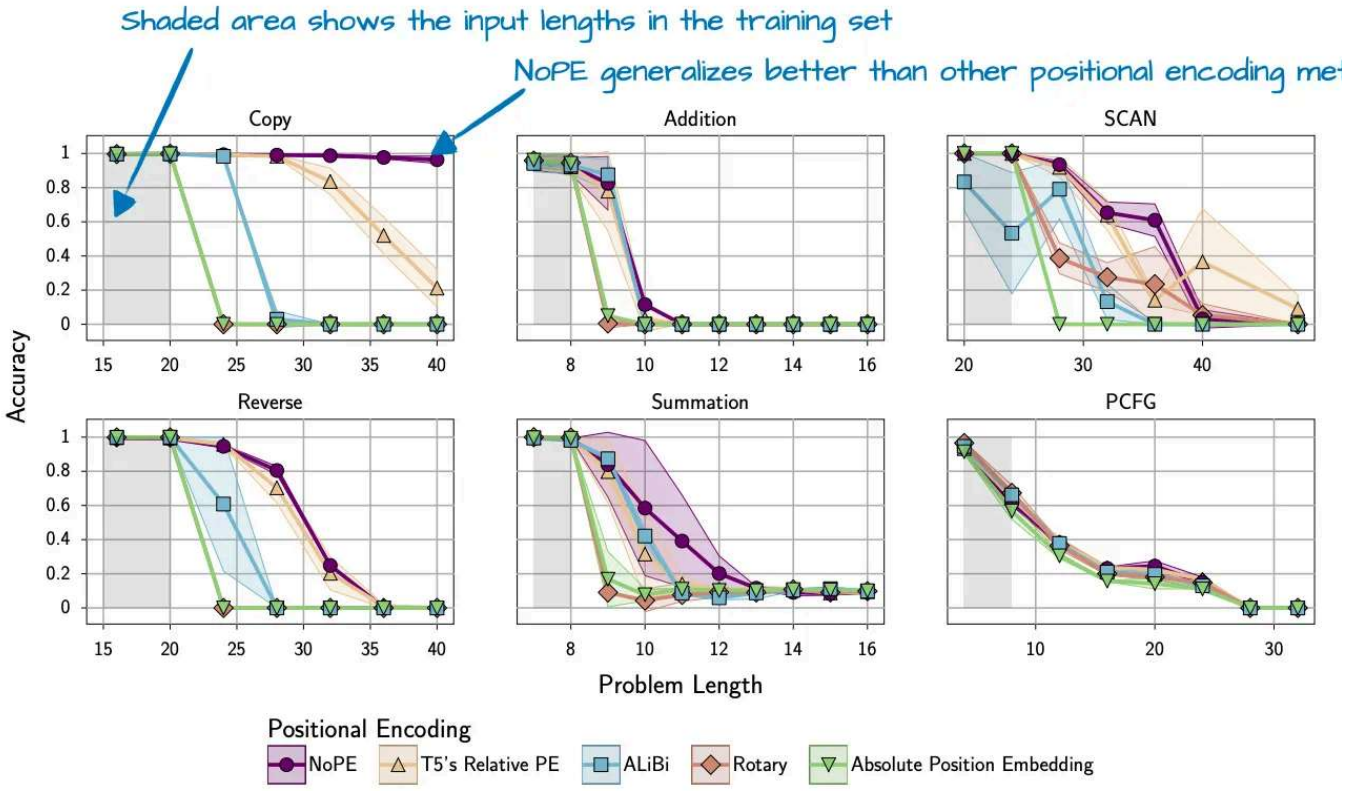


Figure 23: An annotated figure from the NoPE paper (https://arxiv.org/abs/2305.19466) showing better length generalization with NoPE.

图23：来自NoPE论文（https://arxiv.org/abs/2305.19466）的带注释的图，展示了NoPE在长度泛化方面的优势。

Note that the experiments shown above were conducted with a relatively sr GPT-style model of approximately 100 million parameters and relatively sm context sizes. It is unclear how well these findings generalize to larger, contemporary LLMs.

请注意，上述实验是使用一个相对较小的、约有1亿参数的GPT风格模型以及相对转的上下文规模进行的。目前尚不清楚这些发现能在多大程度上推广到更大的当代大模型（LLMs）。

For this reason, the SmolLM3 team likely only "applied" NoPE (or rather om RoPE) in every 4th layer.

出于这个原因，SmolLM3团队可能只在每四层中“应用”一次NoPE（或者更确切地说，省略一次RoPE）。

8. Kimi 2 8.基米2

Kimi 2 recently made big waves in the AI community due to being an open-weight model with an incredibly good performance. According to benchma it's on par with the best proprietary models like Google's Gemini, Anthropic Claude, and OpenAI's ChatGPT models.

觅我2最近在人工智能领域引起了轩然大波，因为它是一款性能卓越的开源大模型。据基准测试，它与谷歌的Gemini、Anthropic的Claude和OpenAI的ChatGPT等最的专有模型不相上下。

A notable aspect is its use of a variant of the relatively new Muon optimizer over AdamW. As far as I know, this is the first time Muon was used over Ada for any production model of this size (previously, it has only been shown to scale up to 16B). This resulted in very nice training loss curves, which probal helped catapult this model to the top of the aforementioned benchmarks.

值得注意的一点是，它使用了相对较新的Muon优化器的一个变体，而非AdamW。我所知，这是首次在如此规模的任何生产模型中使用Muon而非AdamW（此前，它被证明可扩展至160亿参数规模）。这带来了非常理想的训练损失曲线，这可能是该型在上述基准测试中名列前茅的原因之一。

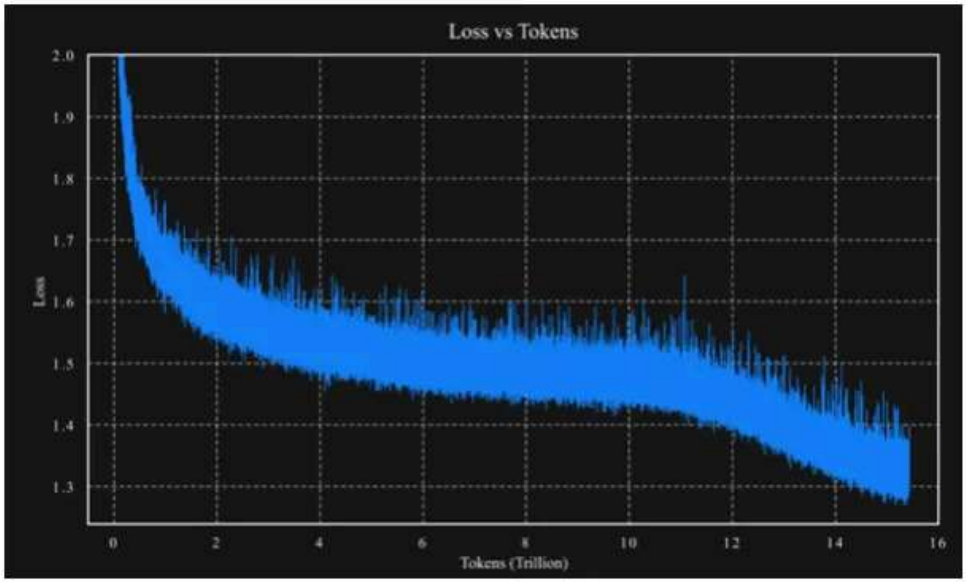
While people commented that the loss was exceptionally smooth (due to th lack of spikes), I think it's not exceptionally smooth (e.g., see the OLMo 2 los curve in the figure below; also, the L2 norm of the gradient would probably a better metric to track training stability). However, what's remarkable is hov well the loss curve decays.

虽然有人评论说损失下降得格外平稳（因为没有峰值），但我认为并非如此（例如下图中OLMo 2的损失曲线；此外，梯度的L2范数可能是跟踪训练稳定性的更好指标）。然而，值得注意的是损失曲线的衰减情况非常好。

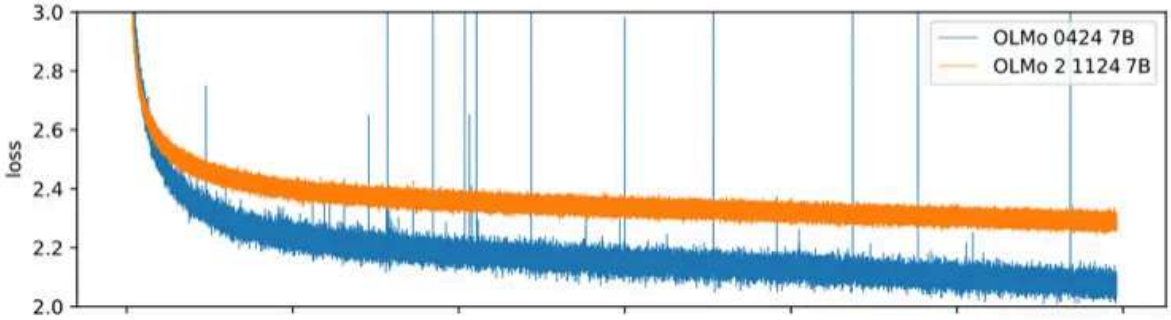
However, as mentioned in the introduction of this article, training methodologies are a topic for another time.

然而，正如本文引言中所提到的，训练方法是另一个时间的话题。

Kimi K2 training loss curve



OLMo & OLMo 2 training loss curves



The model itself is 1 trillion parameters large, which is truly impressive.

该模型本身有1万亿参数，着实令人印象深刻。

It may be the biggest LLM of this generation as of this writing (given the constraints that Llama 4 Behemoth is not released, proprietary LLMs don't count, and Google's 1.6 trillion Switch Transformer is an encoder-decoder architecture from a different generation).

在撰写本文时，它可能是这一代中规模最大的大语言模型（前提是Llama 4 Behem尚未发布，专有大语言模型不算在内，且谷歌的1.6万亿参数的Switch Transforme于另一代的编码器-解码器架构）。

It's also coming full circle as Kimi 2 uses the DeepSeek-V3 architecture we covered at the beginning of this article except they made it larger, as shown

the figure below.

这也算是兜了一圈，因为Kimi 2采用了本文开头介绍的DeepSeek-V3架构，只不过们将其规模扩大了，如下图所示。

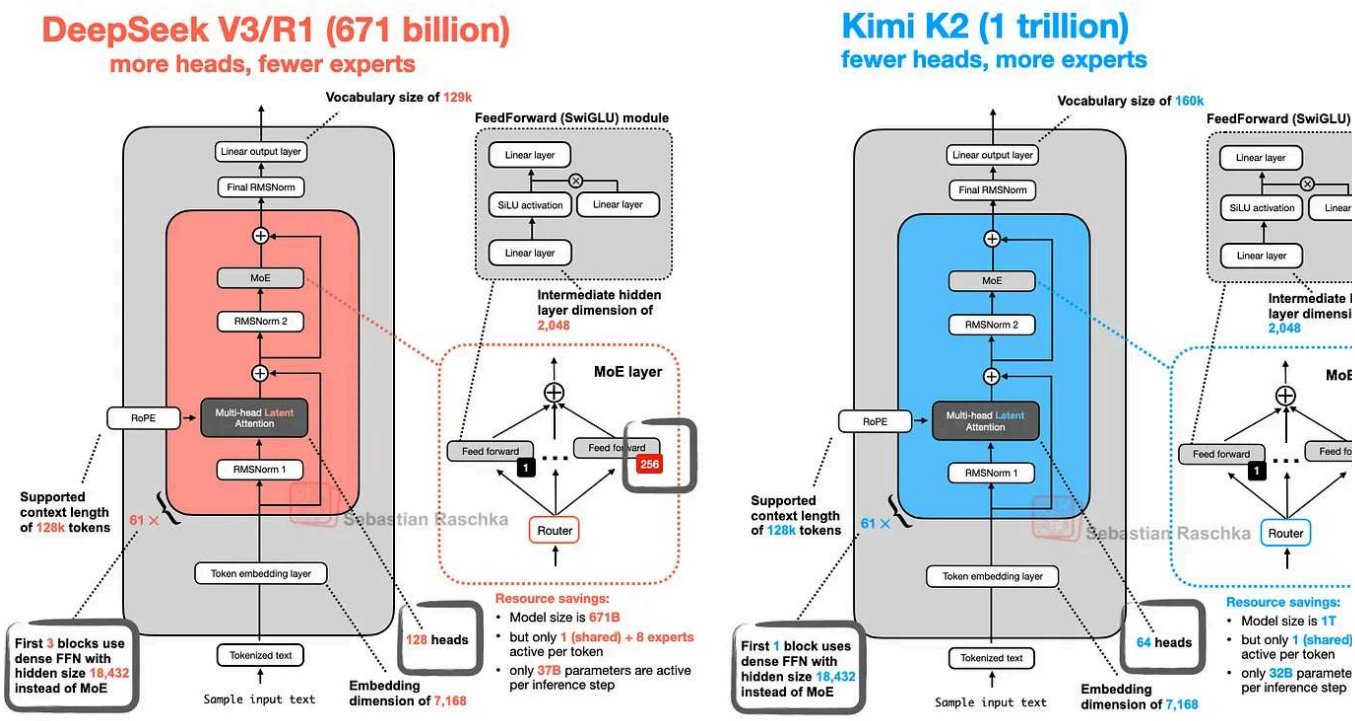


Figure 25: An architecture comparison between DeepSeek V3 and Kimi K2.

图25：DeepSeek V3与Kimi K2的架构对比。

As shown in the figure above, Kimi 2.5 is basically the same as DeepSeek V3 except that it uses more experts in the MoE modules and fewer heads in the Multi-head Latent Attention (MLA) module.

如上图所示，Kimi 2.5与DeepSeek V3基本相同，只是它在混合专家（MoE）模块使用了更多的专家，在多头潜在注意力（MLA）模块中使用了更少的头。

Kimi 2 is not coming out of nowhere. The earlier Kimi 1.5 model discussed i the Kimi k1.5: Scaling Reinforcement Learning with LLMs paper, was impress as well. However, it had the bad luck that the DeepSeek R1 model paper wa published on exactly the same date on January 22nd. Moreover, as far as I know, the Kimi 1.5 weights were never publicly shared.

Kimi 2并非凭空出现。此前在《Kimi k1.5：使用大语言模型扩展强化学习》论文中论的Kimi 1.5模型也令人印象深刻。然而，不巧的是，DeepSeek R1模型的论文在

22日同一天发表。此外，据我所知，Kimi 1.5的权重从未公开分享过。

So, most likely the Kimi K2 team took these lessons to heart and shared Kim as an open-weight model, before DeepSeek R2 was released. As of this writ Kimi K2 is the most impressive open-weight model.

所以，极有可能是 kimi K2 团队吸取了这些教训，在 DeepSeek R2 发布之前，将 kimi K2 作为一款不限权重模型发布。在撰写本文时，kimi K2 是最令人印象深刻的限权重模型。

After all these years, LLM releases remain exciting, and I am curious to s what's next!

这么多年过去了，大语言模型的发布依然令人兴奋，我很期待看到接下来会有什么展！

This magazine is a personal passion project, and your support helps keep it alive. If you would like to contribute, there are a few great ways:

这本杂志是一个源于个人热爱的项目，您的支持让它得以存续。如果您愿意贡献一量，这里有几种很棒的方式：

- ***Grab a copy of my book**. Build a Large Language Model (From Scratch) walks you through building an LLM step by step, from tokenizer to train*

  ***购买我的书**。《从零开始构建大语言模型》将逐步指导你构建一个大语言模型，分词器到训练。*

- ***Check out the video course**. There's now a 17-hour video course based the book, available from Manning. It follows the book closely, section by section, and works well both as a standalone or as a code-along resource. The video course is ad-free (unlike the YouTube version) and has a clear more structured format. It also contains 5 additional hours of pre-requisite video material created by Abhinav Kimothi.*

  ***查看视频课程**。现在有一门基于本书的17小时视频课程，可从曼宁出版社获取，紧密跟随书本内容，逐节讲解，既可以作为独立的学习资源，也适合与代码同步*

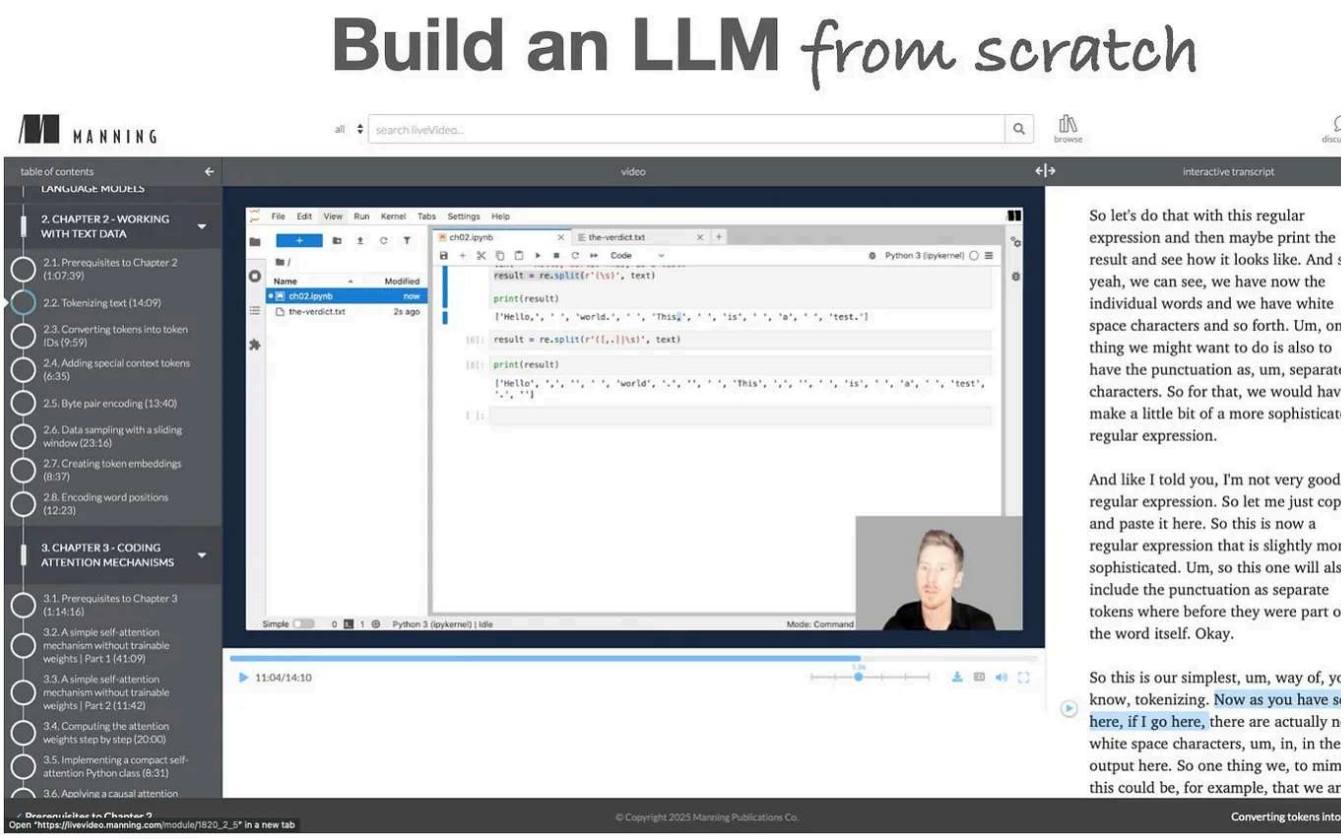
习。该视频课程无广告（与YouTube版本不同），格式更简洁、更有条理。此
它还包含由阿比纳夫·基莫蒂（Abhinav Kimothi）制作的5小时额外的前置视
料。

- ***Subscribe**. A paid subscription helps to make my writing sustainable an*
  gives you access to additional contents.

  ***订阅**。付费订阅有助于让我的写作可持续进行，并让你能够获取更多内容。*

Thanks for reading, and for helping support independent research!

感谢阅读，并感谢您对独立研究的支持！



Subscribe to Ahead of AI 订阅《AI前沿》

By Sebastian Raschka · Hundreds of paid subscribers

作者：塞巴斯蒂安·拉施卡 · 数百名付费订阅者

Ahead of AI specializes in Machine Learning & AI research and is read by tens of thousands researchers and practitioners who want to stay ahead in the ever-evolving field.

《Ahead of AI》专注于机器学习与人工智能研究，受到成千上万希望在这个不断发展的领域保持领先地
研究人员和从业者的关注。

352 Likes 352个赞 · 35 Restacks 35次重新分享

Discussion about this post 关于这篇文章的讨论

Comments 评论 **Restacks 转发**

Write a comment...

Leo Benaharon 利奥·贝纳哈伦 2d 2天前

❤ Liked by Sebastian Raschka, PhD 受到塞巴斯蒂安·拉施卡博士的喜爱

Amazing article! This is evidence that we haven't hit a wall yet with LLMs as all these lab haven't converged to the same architectures.

这篇文章太棒了！这表明我们在大语言模型方面尚未遇到瓶颈，因为所有这些实验室并未采用相构。

Cohere Labs is also doing some great work for open source and have some interesting feel a lot of people don't know who they are as they are trying to appeal to businesses/governments.

Cohere Labs也在为开源领域做一些出色的工作，并且有一些有趣的成果。我觉得很多人不知道司，因为他们主要面向企业和政府开展业务。

LIKE (4) 点赞（4次） REPLY 回复

1 reply by Sebastian Raschka, PhD 塞巴斯蒂安·拉施卡博士的1条回复

Daniel Kleine 丹尼尔·克莱恩 2d 2天前

Liked by Sebastian Raschka, PhD 被塞巴斯蒂安·拉施卡博士点赞

Great overview! 很棒的概述!

As a small side note, I noticed that in Fig. 4, the bottom left comment appears to read 'I Should this perhaps be 'MLA' instead?

顺带提一下，我注意到在图4中，左下方的注释似乎写的是“MQA”。它是不是应该写成“MLA”呢

LIKE (2) 点赞（2） REPLY 回复 S

6 replies by Sebastian Raschka, PhD and others

塞巴斯蒂安•拉施卡博士等人的6条回复

14 more comments... 还有14条评论...