

Arquiteturas de LLMs

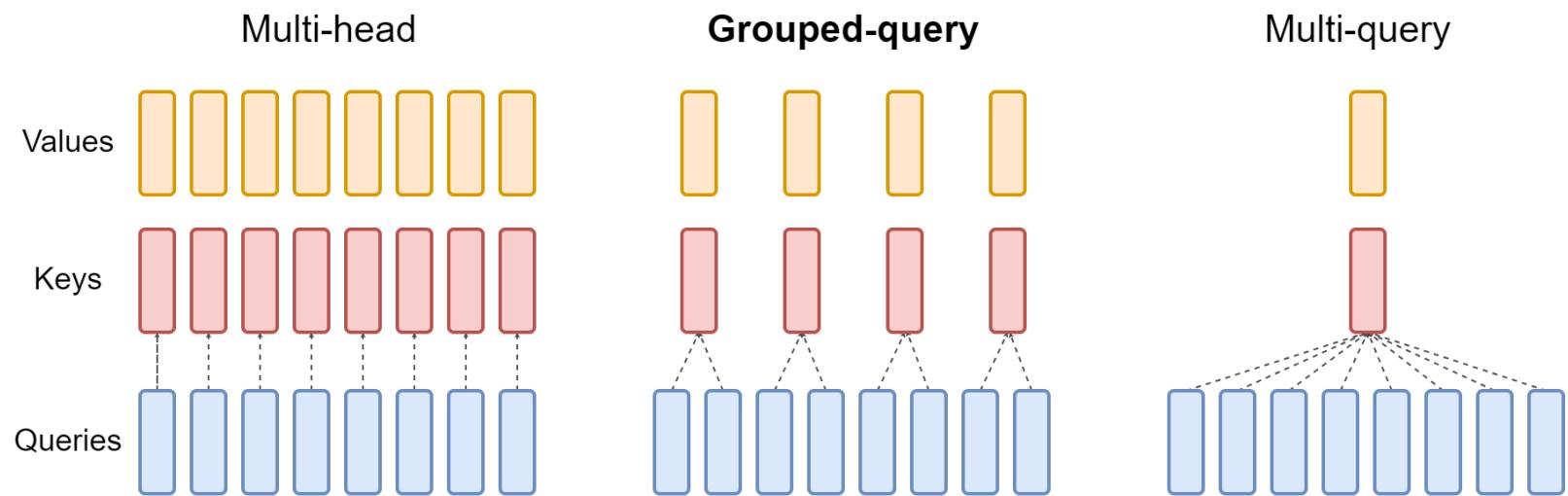
Tópicos em Ciência de Dados

Prof. Dr. Denis Mayr Lima Martins

Pontifícia Universidade Católica de Campinas



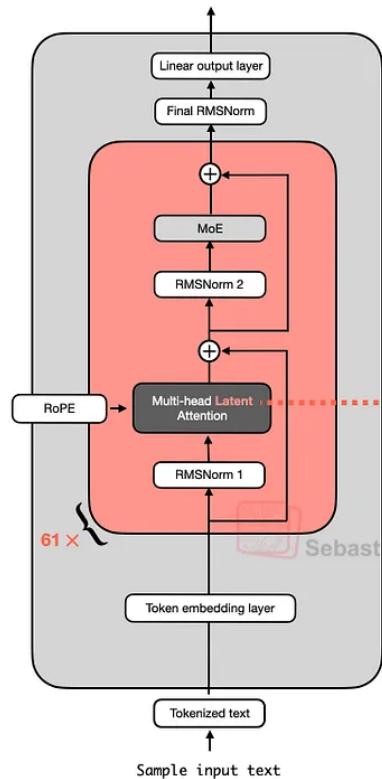
Grouped-Query Attention (GQA)



Comparação MHA (esquerda), GQA (centro) e MQA (direita). Fonte: [GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints](#).

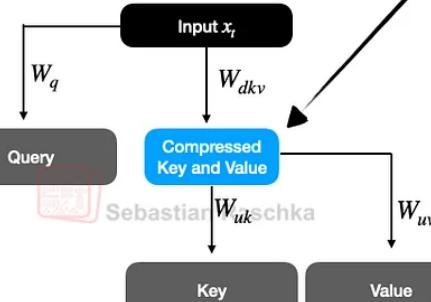
DeepSeek V3/R1

DeepSeek V3/R1



Multi-head Latent Attention (MLA)

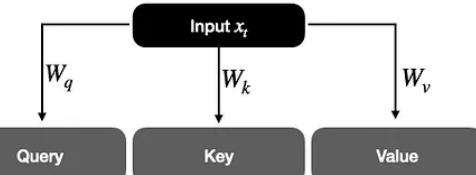
Inference step t :



Key idea: Reduces memory usage in KV cache

Regular Multi-head Attention (MHA)

Inference step t :



Multihead Latent Attention (MLA). Fonte: [Raschka](#).

DeepSeek V3/R1: Resultados de MLA

Unlike what previous papers found, this paper finds that MHA performs much BETTER than GQA

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.

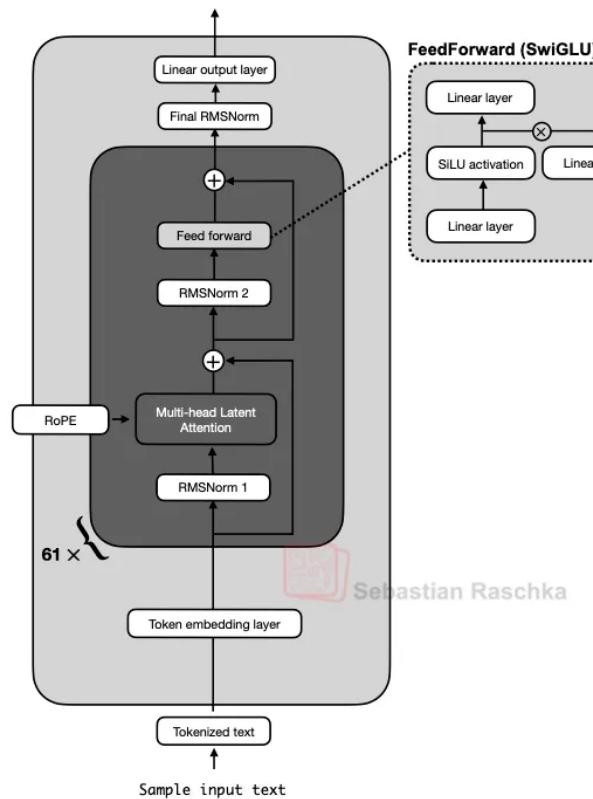
The memory requirements for MLA are much lower than for MHA

MLA performs better than MHA (here tested on Mixture-of-Experts architectures)

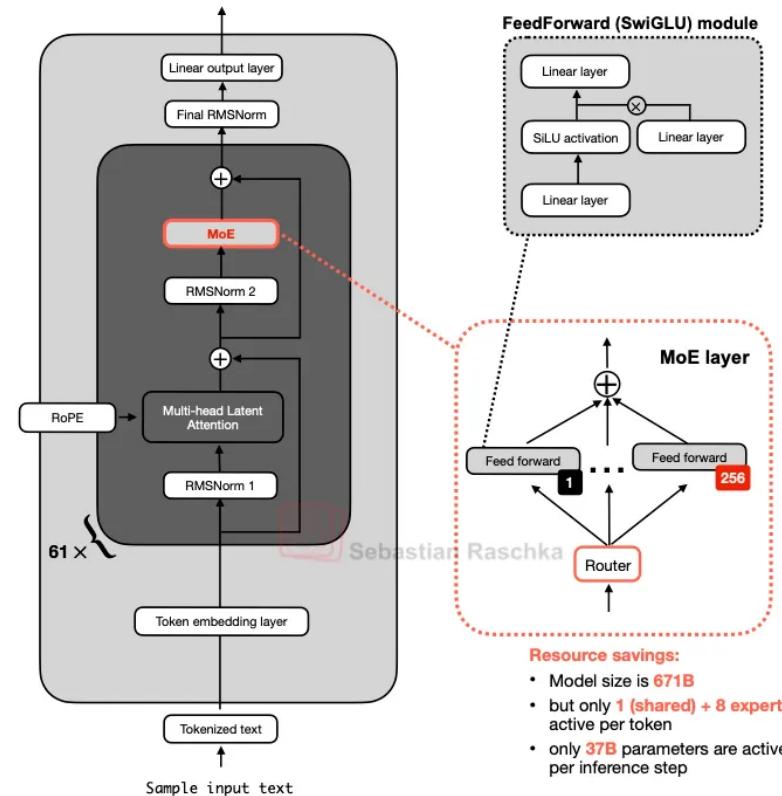
Multihead Latent Attention (MLA). Fonte: [Raschka](#).

Mixture of Experts (MoE)

Architecture
without MoE (“dense”)



DeepSeek V3/R1
with MoE (“sparse”)



- Resource savings:**
- Model size is **671B**
 - but only **1 (shared) + 8 experts** active per token
 - only **37B** parameters are active per inference step

MoE no DeepSeek V3. Fonte: [Raschka](#).

DeepSeek V3/R1



From “post-training” to “training”

DeepSeek V3 used .18% of compute on post-training.

DeepSeek V3 pretraining took <2 months.

DeepSeek R1 RL training took “a few weeks”*

DeepSeek R1 could already be ~10-20% of compute in GPU hours.

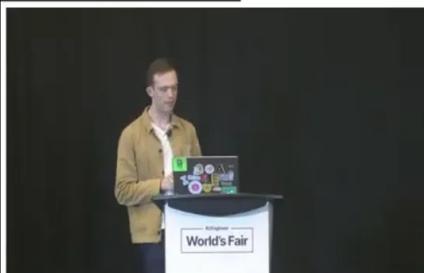
Scaling RL has just begun.

Training Costs	Pre-Training	Context Extension	Post-Training	Total
in H800 GPU Hours	2664K	119K	5K	2788K
in USD	\$5.328M	\$0.238M	\$0.01M	\$5.576M

Training costs of DeepSeek-V3, assuming the rental price of H800 is \$2 per GPU hour.

Lambert | Next Generation Reasoning Models 34

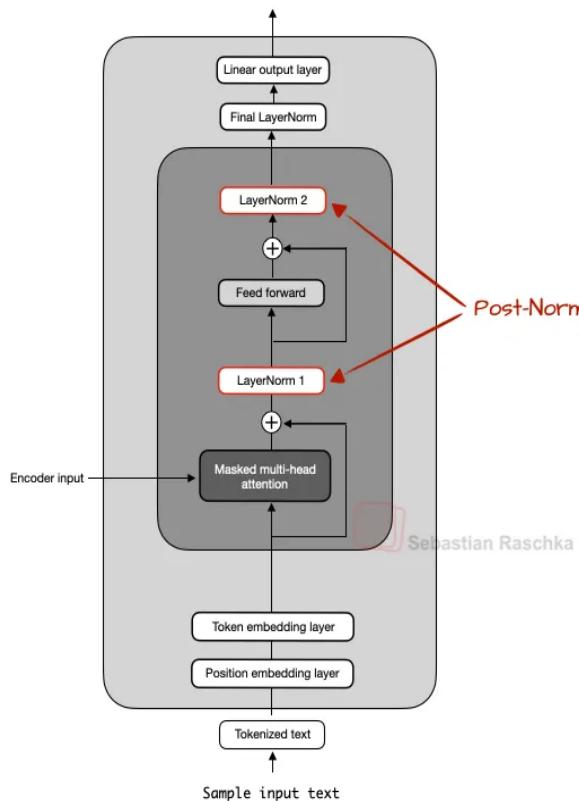
tweet from a DeepSeek employee.



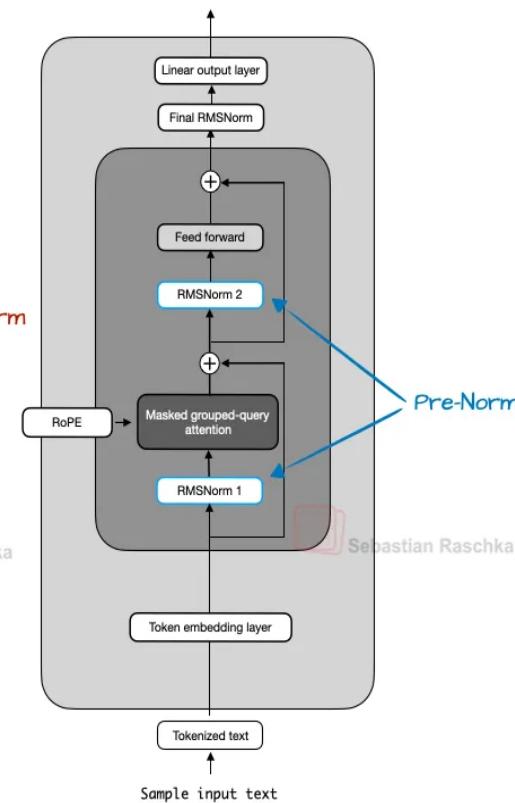
DeepSeek Metrics. Fonte: [Lance Martin](#).

Normalization Layer

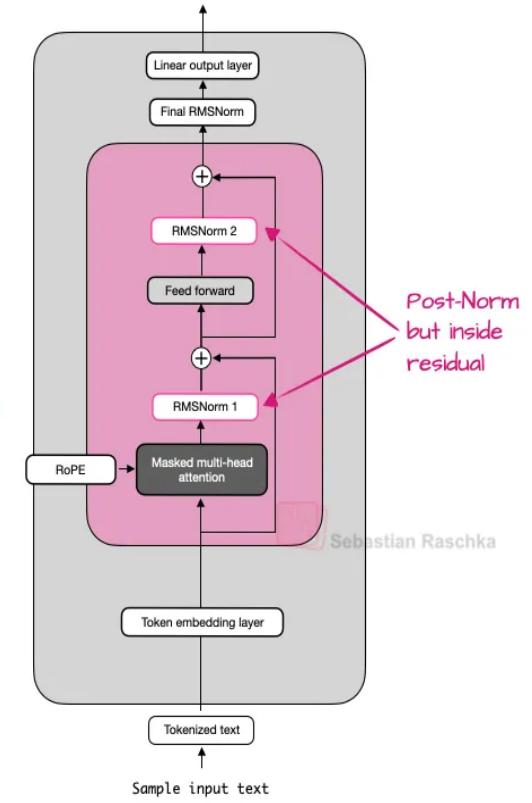
Decoder module of
original Transformer



Llama 3 8B



OLMo 2 7B



Post-Norm x Pre-Norm. Fonte: [Raschka](#).

Root Mean Square Layer Normalization

$\text{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \epsilon}}$, onde \odot é a multiplicação elemento-a-elemento.

```
class RMSNorm(nn.Module):
    def __init__(self, normalized_shape: list | tuple,
                 eps: float = 1e-5, element_affine: bool = True,
                 ):
        super().__init__()
        self.eps = eps
        self.element_affine = element_affine
        if self.element_affine:
            self.gamma = nn.Parameter(torch.ones(normalized_shape))
        else:
            self.register_parameter("gamma", None)

    def forward(self, x: torch.Tensor):
        x = x * torch.rsqrt(self.eps + x.pow(2).mean(dim=-1,
keepdim=True))
        return x if self.gamma is None else x * self.gamma
```

Query-Key Normalization

Ideia: Aplicar L2 norm às matrizes Q e K antes do cálculo de atenção.

De: $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ Para:

$$\text{QK-Norm}(Q, K, V) = \text{softmax}\left(\frac{Q}{\|Q\|_2} \cdot \left(\frac{K}{\|K\|_2}\right)^T\right)V$$

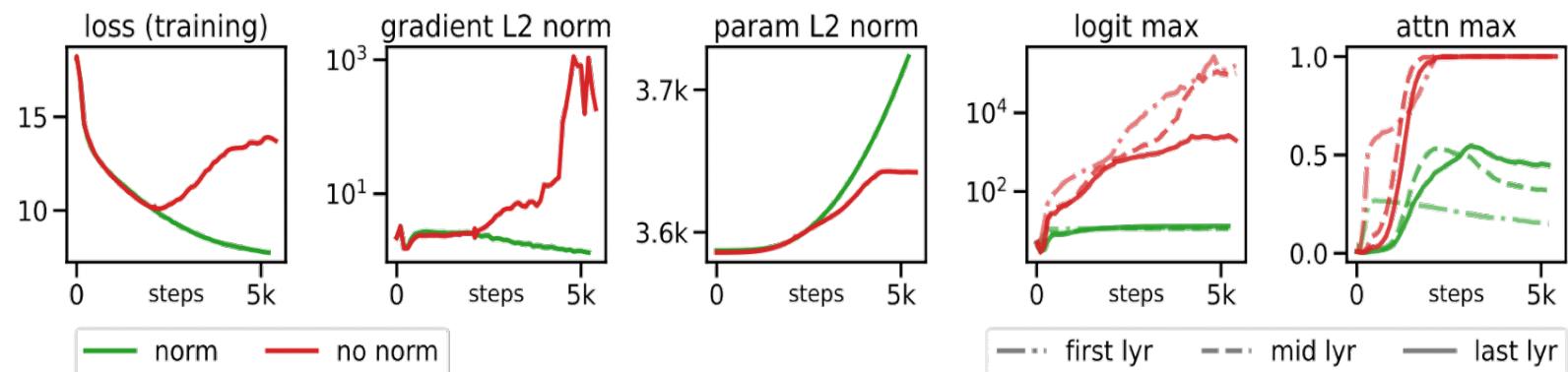
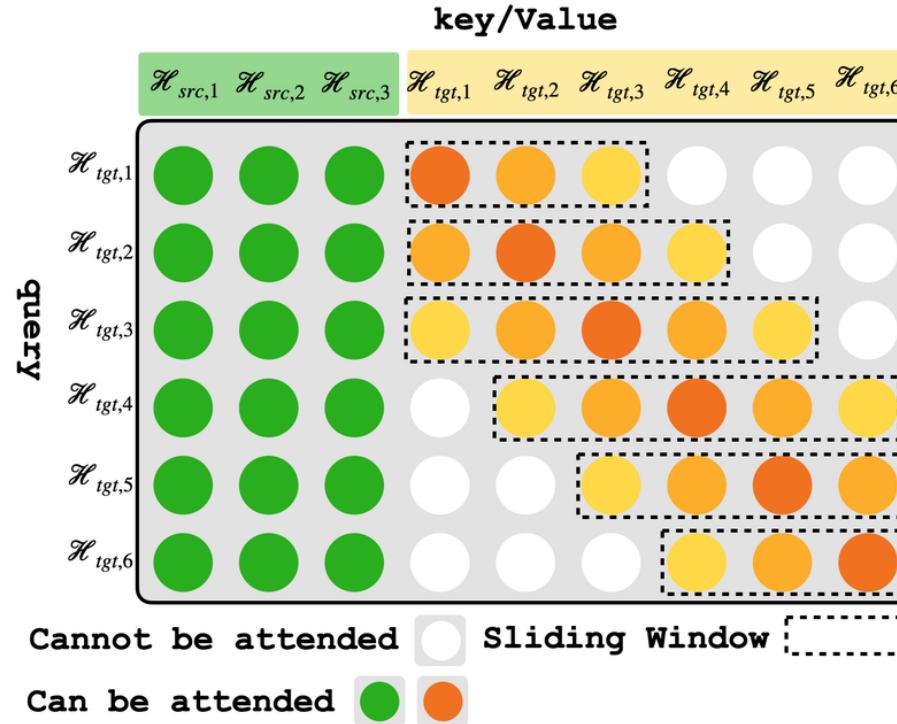


Figure 1: Effect of query/key normalization on an 8B parameter model.

Resultados de QK-Norm. Fonte: <https://arxiv.org/abs/2302.05442>.

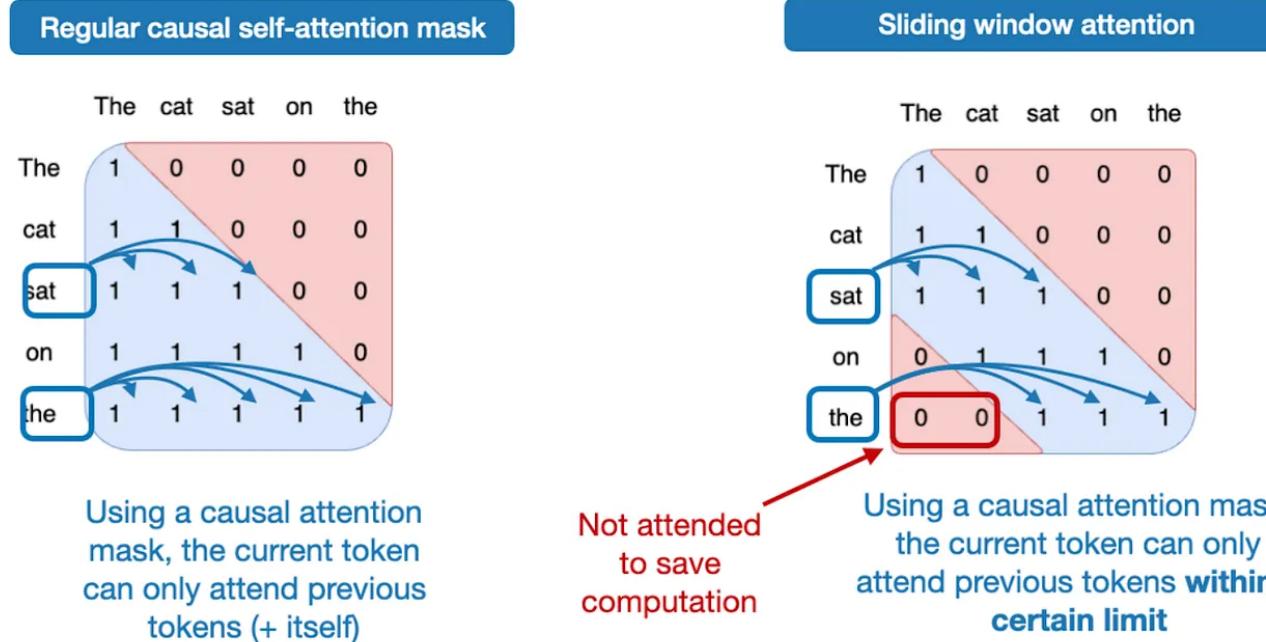
Sliding Window Attention (SWA)



Exemplo de SWA. Fonte: [Research Gate](#).

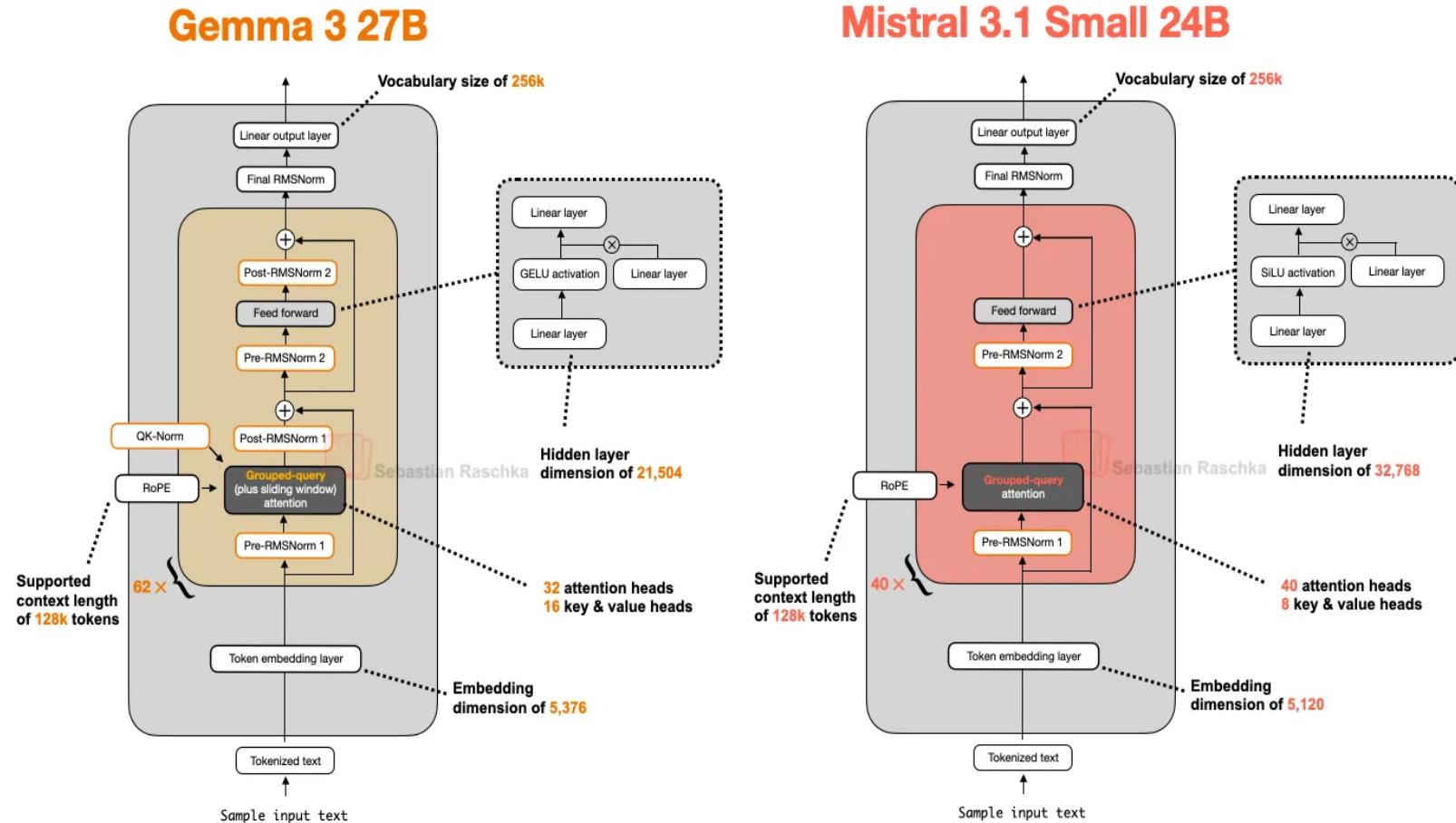
Detalhes em: <https://arxiv.org/html/2502.18845v2>

Sliding Window Attention (SWA)



Comparação Causal Attention x SWA. Fonte: Raschka.

Gemma 3 27B e Mistral 3.1



Comparação entre Gemma 3 27B e Mistral 3.1 Small 24B. Fonte: [Raschka](#).

RoPE

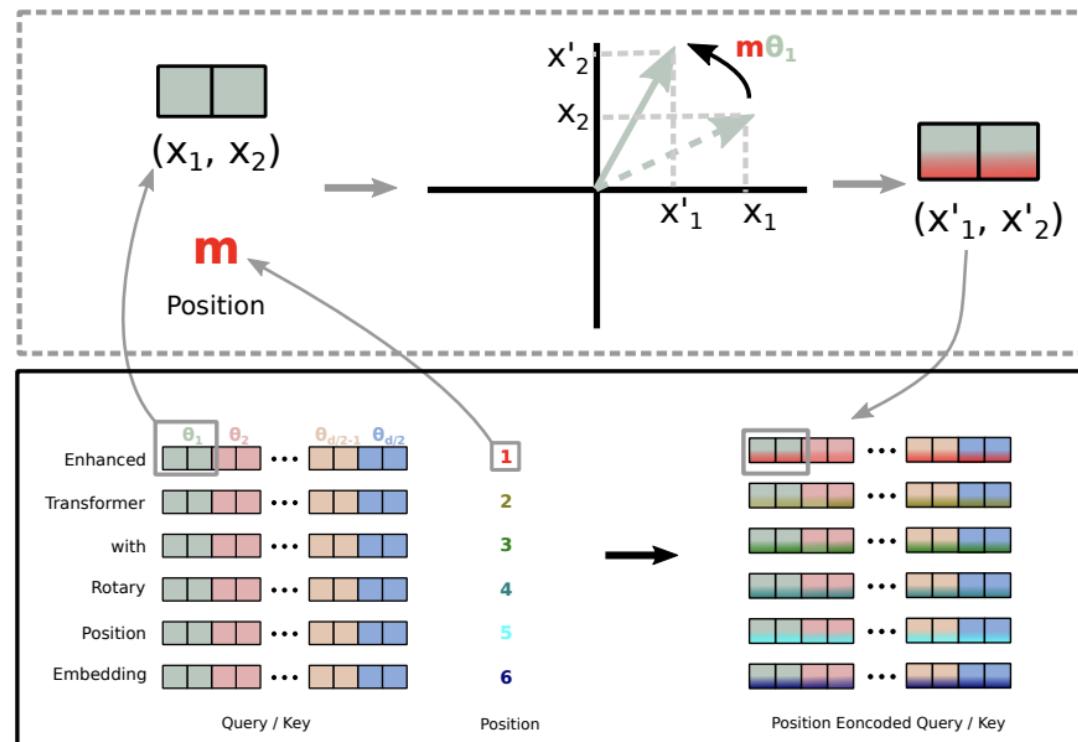
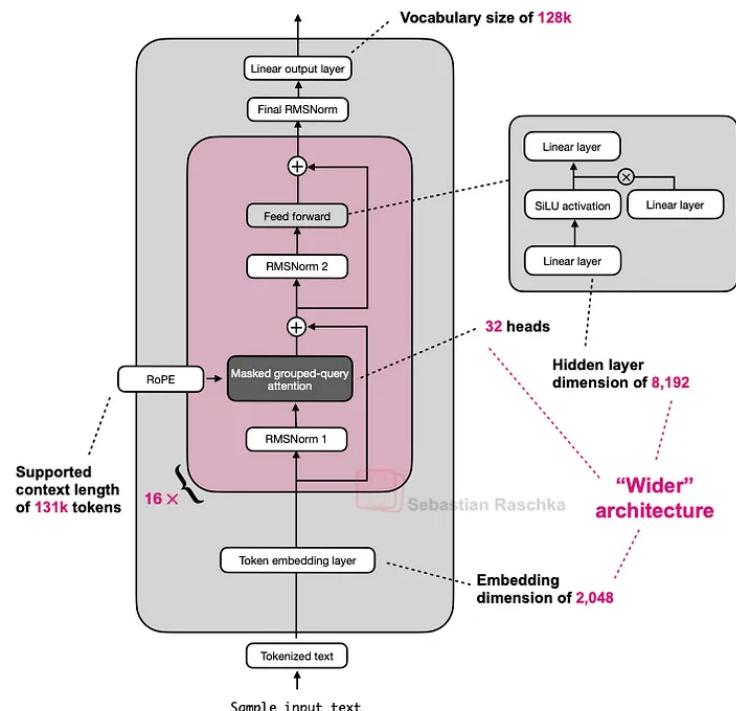


Figure 1: Implementation of Rotary Position Embedding(RoPE).

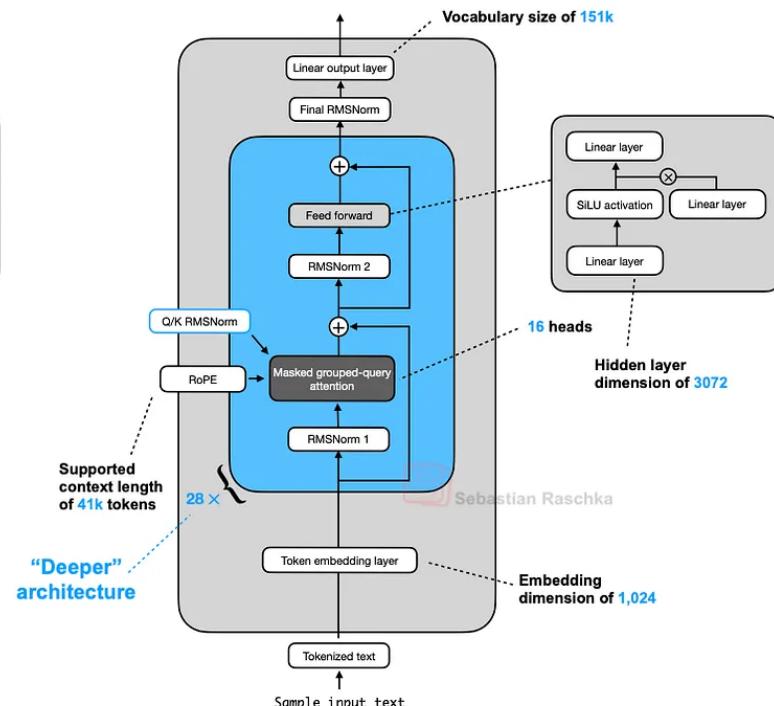
Rotary Positional Embedding. Fonte: <https://arxiv.org/abs/2104.09864>.

Qwen3 0.6B

Llama 3.2 1B



Qwen3 0.6 B



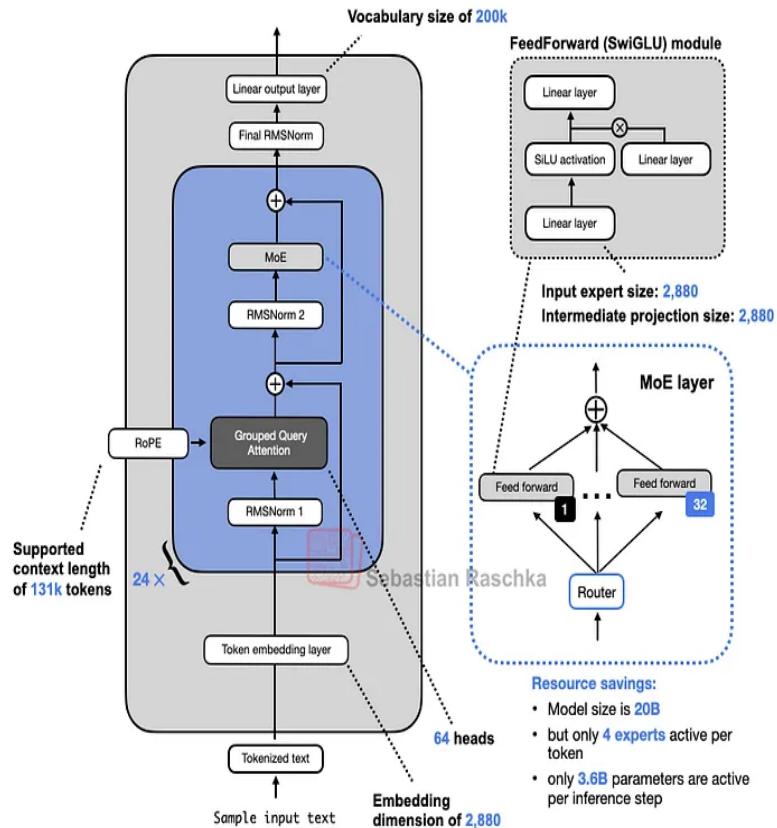
	Tokens/sec	Memory
Llama3Model 1B	42	2.91 GB
Llama3Model 1B compiled	170	3.12 GB

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

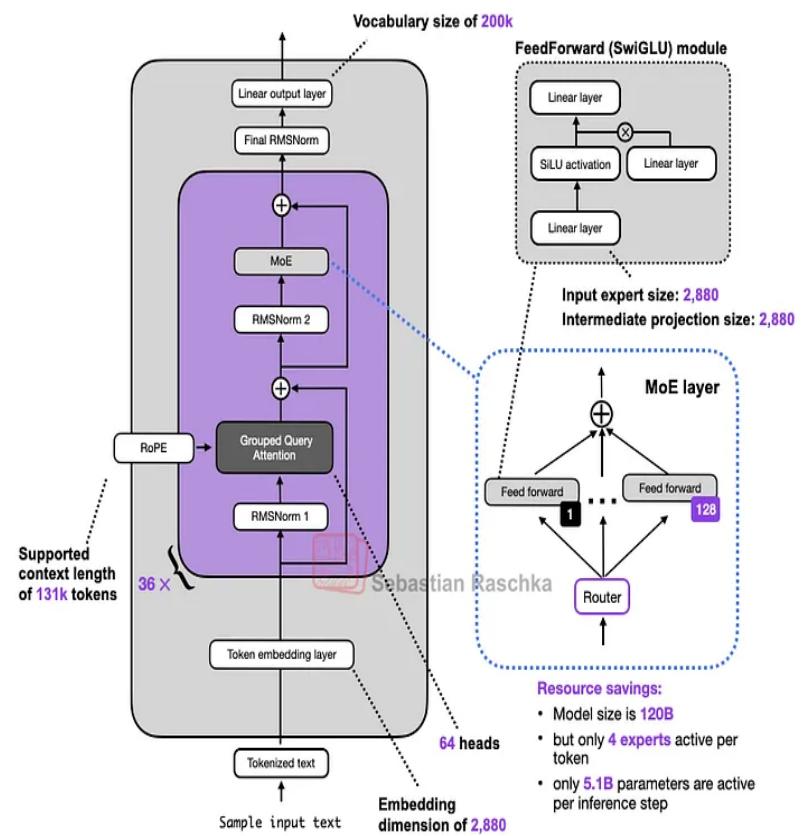
Comparação entre Llama 3.2 1B e Qwen3 0.6B. Fonte: [Raschka](#).

GPT-OSS

GPT-OSS 20B



GPT-OSS 120B



Comparação modelos GPT-OSS. Fonte: [Raschka](#).

Extra: O preço da formatação

Formatted Code

Tokens: **60**

```
public class HelloWorld {\n    public static void main(String[] args) {\n        System.out.println("Hello, World!");\n        int a = 10;\n        int b = 20;\n        int sum = a + b;\n\n        System.out.println("Sum: " + sum);\n    }\n}
```



Unformatted Code

Tokens: **47**

```
public class HelloWorld{public static void\nmain(String[]args){System.out.println("Hello,\nWorld!");int a=10;int b=20;int\nsum=a+b;System.out.println("Sum: "+sum);}}
```

Comparação de tokens no tokenizer GPT-4o. Formatação de código no prompt tem seu preço. Fonte: <https://arxiv.org/html/2508.13666v1>.

Leitura Recomendada

- Sebastian Raschka blog (Ahead of AI):
<https://magazine.sebastianraschka.com/p/the-big-l1m-architecture-comparison>
- The Transformer++: <https://www.gleech.org/tplus>
- Applied LLMs: <https://applied-l1ms.org/>
- Eugene Yan blog: <https://eugeneyan.com/>
- Hugging Face Papers (antigo Papers with Code):
<https://huggingface.co/papers/trending>