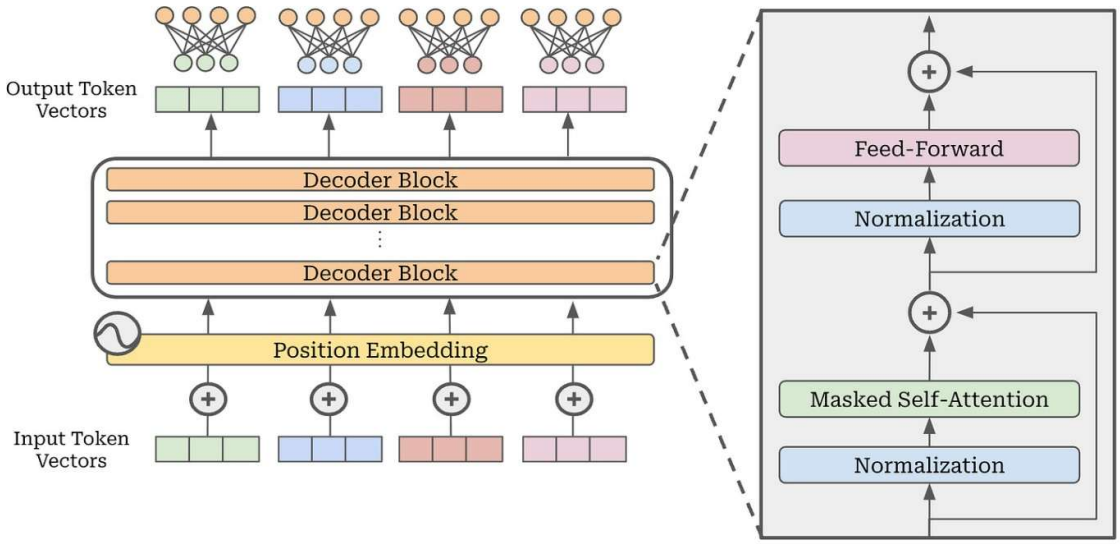


- GPT-oss models, these vectors have a fixed dimension of 2,880, and this same embedding dimension is maintained through every layer of the LLM.
- The decoder-only architecture is comprised of repeated decoder blocks
- The GPT-oss models adopt a pre-normalization structure, which is the most common choice in current LLM architectures
- This means that the normalization layers in the decoder block are placed before both the attention and feed-forward layers, yielding the following structure:

- `Decoder Block Input → Normalization → Masked Self-Attention → Residual Connection → Normalization → Feed-Forward Network → Residual Connection → Decoder Block Output`
- there is no clear answer in terms of whether pre or post-normalization is superior.

Standard pre-normalization structure causes loss spikes in pretraining!

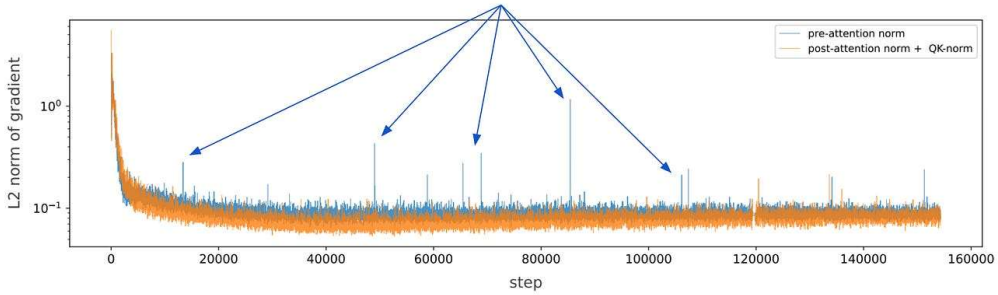
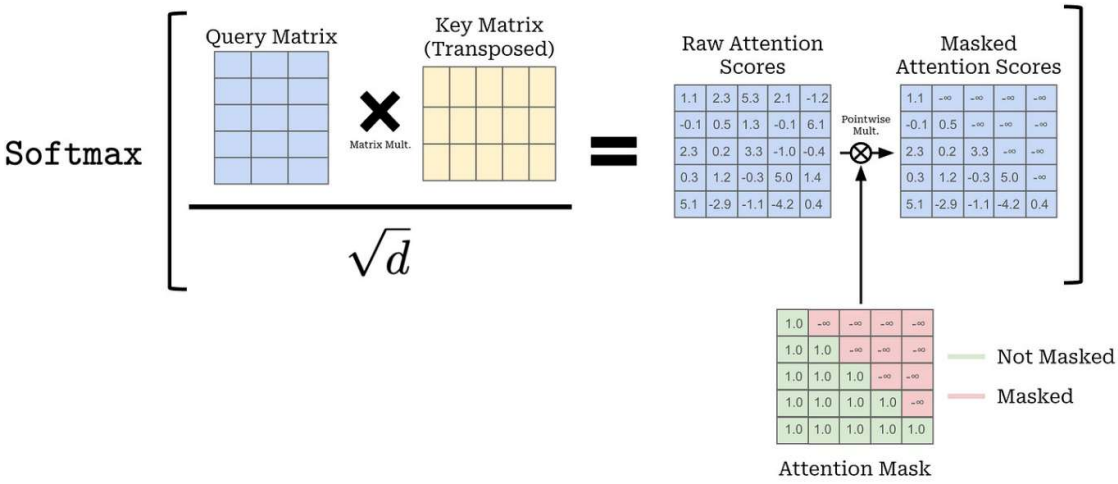


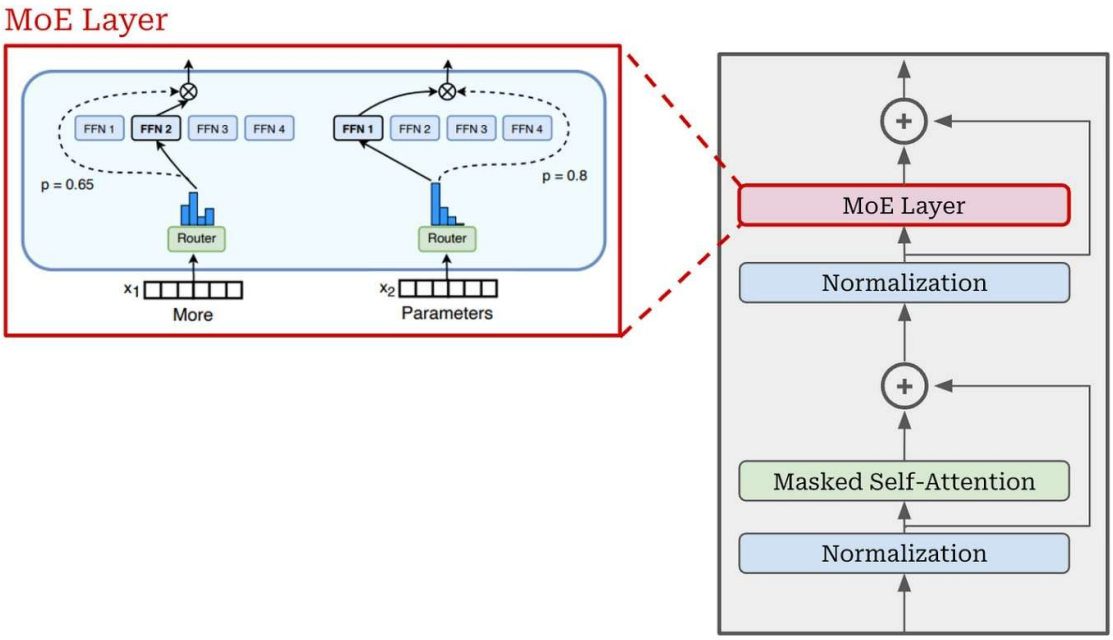
Figure 7 Applying layer norm after the attention and feedforward layers along with a QK-norm improves stability compared to a more standard pre-attention layer norm. These changes reduce the spike score of the gradients from 0.108 to 0.069 when applied together.

- GPT-oss models, each self-attention layer has 64 parallel attention heads. Each of these attention heads use vectors with a dimension of 64, meaning that the key, query and value projections (shown above) transform embedding vectors from a size of 2,880 to 64.
  GPT-oss uses group sizes of eight



  Computing self-attention has quadratic—*or* O(S^2) *where* S *is the sequence length*—complexity.
  sliding window attention
  Mixture-of-Experts (MoE)

- Both GPT-oss models use a Mixture-of-Experts (MoE) architecture. Compared to the decoder-only architecture.



- MoE creates several feed-forward networks, *each with their own independent weights*
- *Not all but every* P-*th layer in the transformer is converted into an MoE layer*.
- We do not want every token to go to all experts as its compuxtationally expensive
- Each token vector->linear layer->produces scores for all N experts and choose top k experts and send tokens to that only.
- Combine output by weighted averaging
- Compute cost per token is low
- That leads to some parameters are active while others are inactive
- GPT-oss, the 20b and 120b models have 32 and 128 total experts within each of their MoE layers.only 4- active for each token

Component	120b	20b
MLP	114.71B	19.12B
Attention	0.96B	0.64B
Embed + Unembed	1.16B	1.16B
Active Parameters	5.13B	3.61B
Total Parameters	116.83B	20.91B
Checkpoint Size	60.8GiB	12.8GiB

Table 1: *Model parameter counts.* We refer to the models as "120b" and "20b" for simplicity, though they technically have 116.8B and 20.9B parameters, respectively. Unembedding parameters are counted towards active, but not embeddings.

- 109b parameter Llama-4 model has 17b active parameters. However, this high sparsity level of GPT-oss is common among the best open-source LLMs:
- DeepSeek-R1 [10] has 671b total parameters and 37b active parameters.
- Qwen-3 [11] MoE models have 30b total parameters and 3b active parameters or 235b total and 22b active parameters.
- *routing collapse*-the model will quickly learn to route all tokens to a single expert. Additionally, MoEs are more likely to experience numerical instabilities during training, potentially leading to a divergence in the training loss;

- Assigns equal probability to all experts in the router.
- Dispatches an equal number of tokens to each expert.
- Fixed capacity factor for every expert, which defines the maximum number of tokens that can be routed to an expert at once. Any tokens that go beyond this capacity factor will simply be dropped

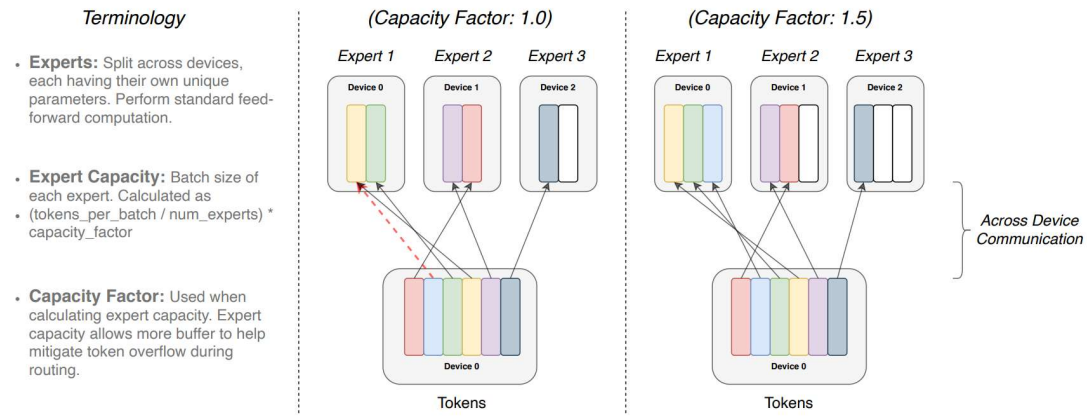
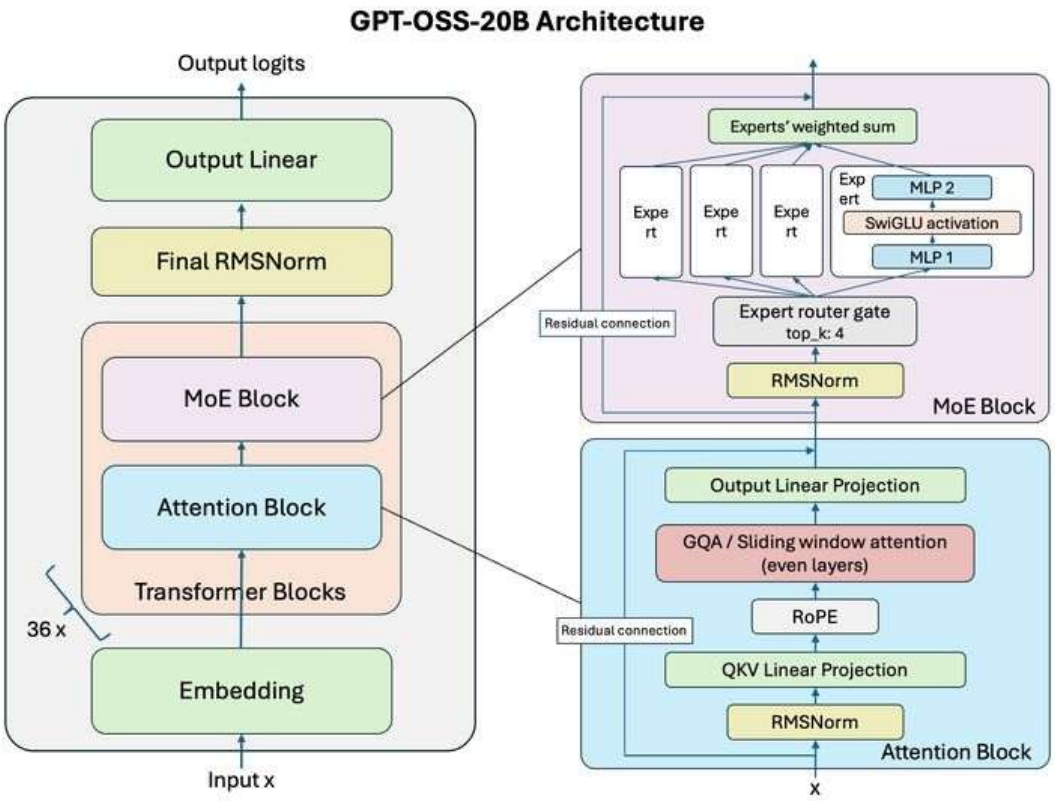
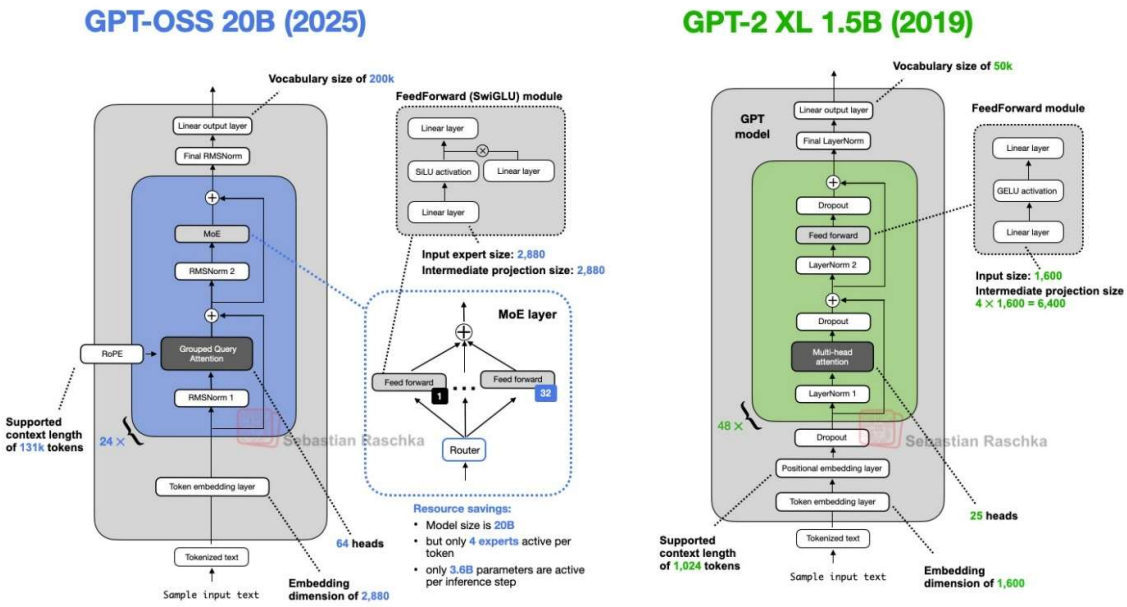


Figure 3: Illustration of token routing dynamics. Each expert processes a fixed batch-size of tokens modulated by the *capacity factor*. Each token is routed to the expert with the highest router probability, but each expert has a fixed batch size of (total_tokens / num_experts) × capacity_factor. If the tokens are unevenly dispatched then certain experts will overflow (denoted by dotted red lines), resulting in these tokens not being processed by this layer. A larger capacity factor alleviates this overflow issue, but also increases computation and communication costs (depicted by padded white/empty slots).

-
- Auxiliary losses-Add small bias term to the logit predicted by roouter to avoid it.

GPT 2 vs GPT-oss

- 

- 

GPT-OSS Architecture Overview

GPT-OSS models are built on a shared architectural foundation:

- **36 Transformer Blocks**: These are stacked sequentially to enable deep reasoning.

- **Grouped Query Attention (GQA)**: Instead of using separate key/value heads for each attention head, GPT-OSS uses 64 attention heads but shares just 8 key/value heads across them. This design significantly reduces memory usage while maintaining attention diversity.
- **Mixture-of-Experts (MoE)**: Each block includes 128 expert MLPs, but only the top 4 are activated per token. This allows the model to benefit from a large parameter space without incurring full computational cost.
- **Rotary Position Embeddings (RoPE)**: Enhanced with YaRN scaling and NTK-by-parts interpolation, RoPE enables the model to handle extended context lengths—scaling from 4K to over 131K tokens—without losing coherence.
- **Sliding-Window Attention & Learned Attention Sinks**: These mechanisms help the model focus computational resources on relevant parts of the input, improving stability and reasoning performance.
- **MXFP4 Quantization**: Expert weights are stored in 4-bit blocks with per-block scaling, allowing even the 120B-parameter model to run on hardware with limited memory.

Data Flow Summary

The model processes input as follows:

Tokens → Embedding → [×36 Transformer Blocks] → Final RMSNorm → Unembedding → Output Logits

Each transformer block follows a consistent pattern:

RMSNorm → Attention → Residual Add → RMSNorm → MoE MLP → Residual Add

This rhythm ensures stable training and efficient computation.

Model Capabilities

GPT-OSS is designed to be:

- **Powerful**: Capable of reasoning, coding, and following instructions across tools.
- **Efficient**: Optimized for modern GPUs, avoiding the need for massive infrastructure.
- **Adaptable**: Easily fine-tuned for specialized tasks, from customer support to scientific research.

Tokenization & Embedding Layer

Before processing text, GPT-OSS uses a tokenizer adapted from OpenAI's o200k, customized for Harmony chat format. This tokenizer:

- Recognizes roles (system, developer, user)
- Supports tool calls and structured messages
- Includes ~201,088 tokens covering common words, special symbols, and protocol-specific tokens

Embedding Details:

- **Embedding Size**: 2880
- **Matrix Shape**: (201,088 × 2880)
- **Output Shape**: (batch_size, seq_len, 2880)

Unlike older models, GPT-OSS applies RoPE inside the attention mechanism, keeping the embedding layer focused purely on semantic content.

Transformer Block Design

Each block acts as a modular reasoning unit, processing token representations through:

1. **RMSNorm**: Normalizes inputs using root mean square, offering computational efficiency and stability.
2. **Q/K/V Projections**: Inputs are transformed into queries, keys, and values. GQA reduces memory usage by sharing key/value heads.
3. **RoPE**: Positional information is injected directly into Q/K vectors, enabling long-context reasoning.
4. **Scaled Dot-Product Attention**: Implemented with Triton kernels, includes causal masking, sliding-window attention (128-token window), and learned sinks for stability.
5. **Output Projection**: Combines multi-head outputs into a unified representation.
6. **Residual Connections**: Merge sub-layer outputs back into the token stream

- **Removing Dropout**
  dropout is rarely used in modern LLMs, and most models after GPT-2 have dropped it (no pun intended).

- **RoPE Replaces Absolute Positional Embeddings**

$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{model}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{model}}}\right) & \text{if } i \text{ is odd} \end{cases}$$

$$\text{softmax}\left(\frac{q_m^T k_n}{\sqrt{|D|}}\right)$$

- The angle component depends only on the position of the vector in the sequence and is independent of the actual token embeddings.
- The token similarity is independent of the angle.

- Instead of adding separate positional encodings, RoPE rotates the query and key vectors based on their position in the sequence.

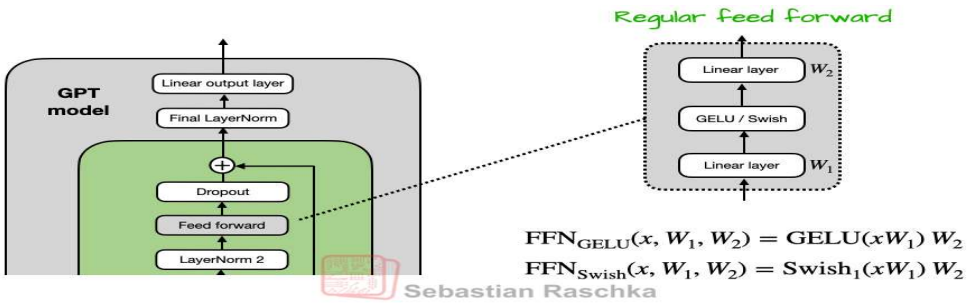
- The rotation preserves the magnitude (maintaining token similarity) while encoding positional information in the angle.
- *This approach allows RoPE to seamlessly integrate both token and positional information into a single operation, making it more efficient and potentially more effective than traditional positional encoding methods.*
- https://medium.com/thedeephub/positional-encoding-explained-a-deep-dive-into-transformer-pe-65cfe8cfe10b

Swish/SwiGLU Replaces GELU

Early GPT architectures used GELU. Why now use Swish over GELU? Swish (also referred to as sigmoid linear unit or SiLU) is considered computationally slightly cheaper.

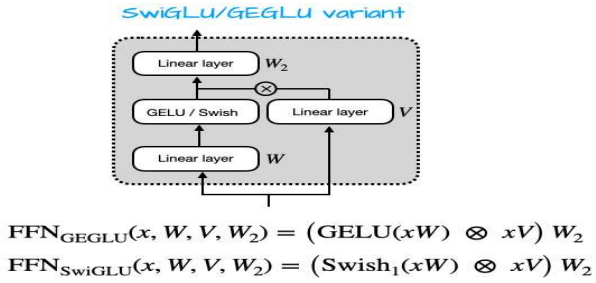
Activation	Formula	Blocks Negatives?	Smooth?
ReLU	$\max(0, x)$	Yes	No
GELU	$\frac{1}{2}x(1 + \text{erf}(x/\sqrt{2}))$	No	Yes
Swish	$x \cdot \sigma(x)$	No	Yes
SwiGLU	$(W_u x) \cdot \text{Swish}(W_v x)$	No	Yes

GELU, which is defined as 0.5x * [1 + erf(x / sqrt(2))]. Here, erf (short for error function) is the integral of a Gaussian and it is computed using polynomial approximations of the Gaussian integral, which makes it more computationally expensive .



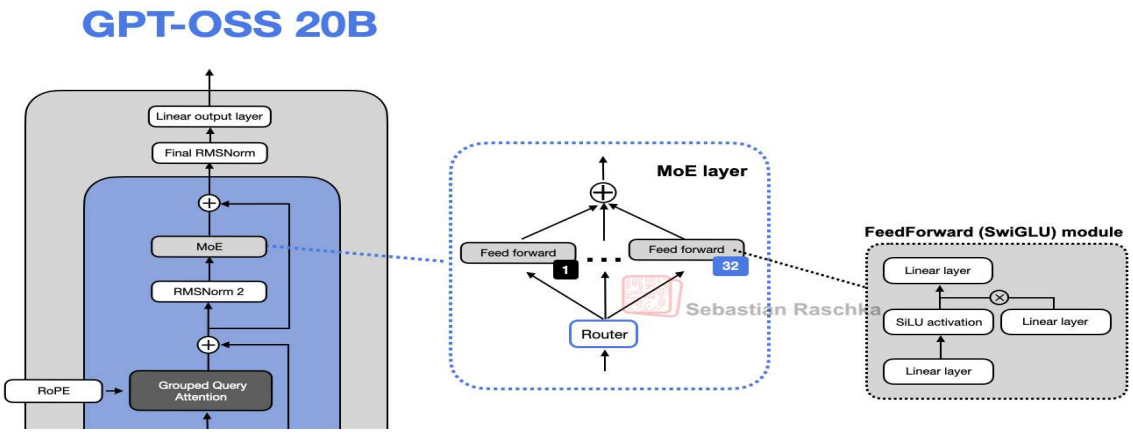
	Score Average
FFN_{ReLU}	72.76
FFN_{GELU}	72.98
$\text{FFN}_{\text{Swish}}$	72.40
FFN_{GLU}	73.95
$\text{FFN}_{\text{GEGLU}}$	73.96
$\text{FFN}_{\text{Bilinear}}$	73.81
$\text{FFN}_{\text{SwiGLU}}$	**74.56**
$\text{FFN}_{\text{ReGLU}}$	73.66
[Raffel et al., 2019]	71.36
ibid. stddev.	0.416

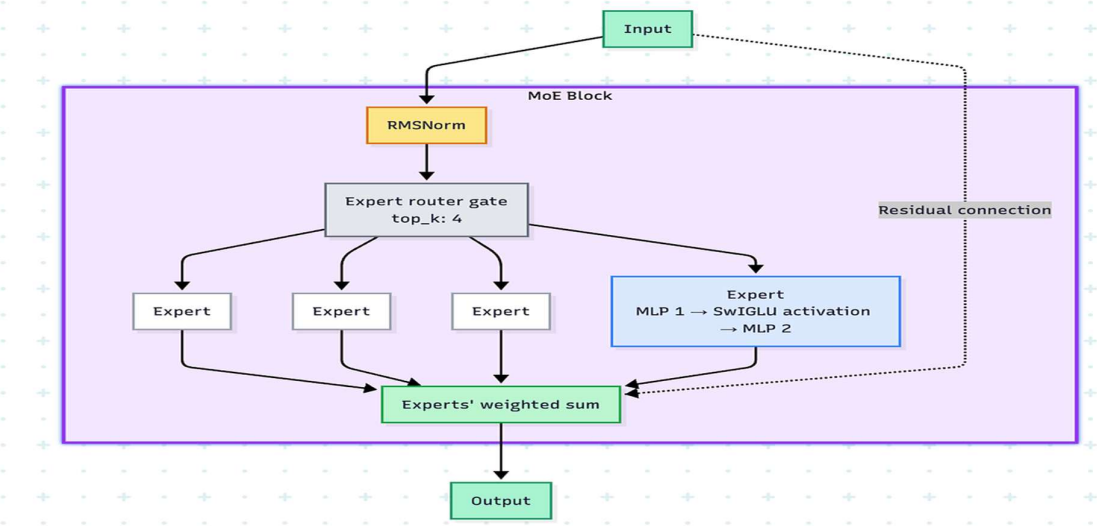
GLU Variants Improve Transformer, https://arxiv.org/abs/2002.05202



So, overall, using the GLU variants results in fewer parameters, and they perform better as well. The reason for this better performance is that these GLU variants provide an additional multiplicative interaction, which improves expressivity (the same reason deep & slim neural nets perform better than shallow & wide neural nets, provided they are trained well).

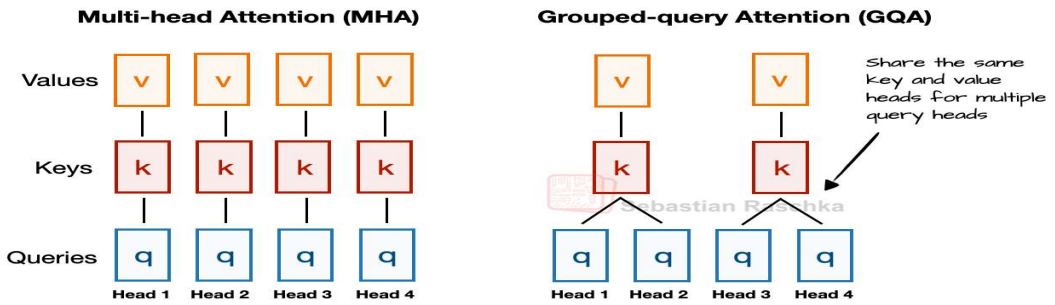
Mixture-of-Experts Replaces Single FeedForward Module





gpt-oss replaces the single feed forward module with multiple feed forward modules, using only a subset for each token generation step. This approach is known as a Mixture-of-Experts (MoE). Increases the model's total parameter count. 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

Grouped Query Attention Replaces Multi-Head Attention

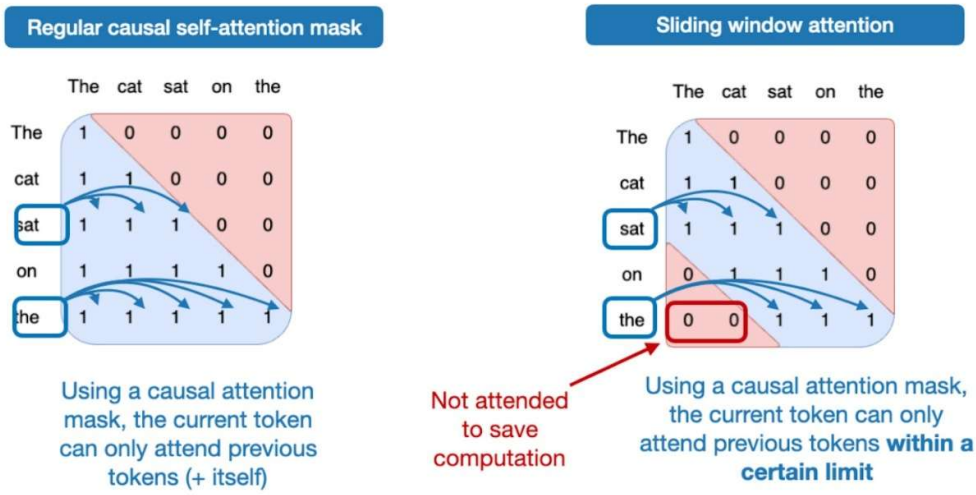


the core idea behind GQA is to reduce the number of key and value heads by sharing them across multiple query heads.

GPT-OSS uses **Grouped Query Attention (GQA)**:

- **64 attention heads total**
- **8 key/value heads** shared across groups
- **head_dim = 64**
- This reduces the memory footprint of K/V caches without sacrificing diversity in attention patterns.

Sliding Window Attention



limiting the window over which self-attention is computed.

The GPT-oss models replace every other masked self-attention module (i.e., a 1:1 ratio) with sliding window attention. The window size used in GPT-oss is 128 tokens, which is small compared to other models; e.g., Gemma-2 and 3 use window sizes of 4K and 1K tokens, respectively

Each attention head has a learned bias in the denominator of the softmax, similar to off-by-one attention and attention sinks, which enables the attention mechanism to pay no attention to any tokens

The models use alternating dense and locally banded sparse attention patterns, similar to GPT-3.

- ***Use dense layers to learn global relationships.***
- ***Use sparse layers to reduce compute and focus on local structure.***
- ***Achieve a balance between performance and efficiency.***

RMSNorm Replaces LayerNorm

Both LayerNorm and RMSNorm stabilize activation scales and improve optimization, but RMSNorm is often preferred in large-scale LLMs because it is cheaper to compute. Unlike LayerNorm, RMSNorm has no bias (shift) term and reduces the expensive mean and variance computations to a single root-mean-square operation. This reduces the number of cross-feature reductions from two to one, which lowers communication overhead on GPUs and improving training efficiency.

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

- x = input vector (e.g., token representation)
- μ = mean of all elements in the vector
- $\sigma 2$ = variance of all elements
- γ, β = learnable scale and shift parameters
- ϵ = small constant for stability

$$\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)} \cdot \gamma$$

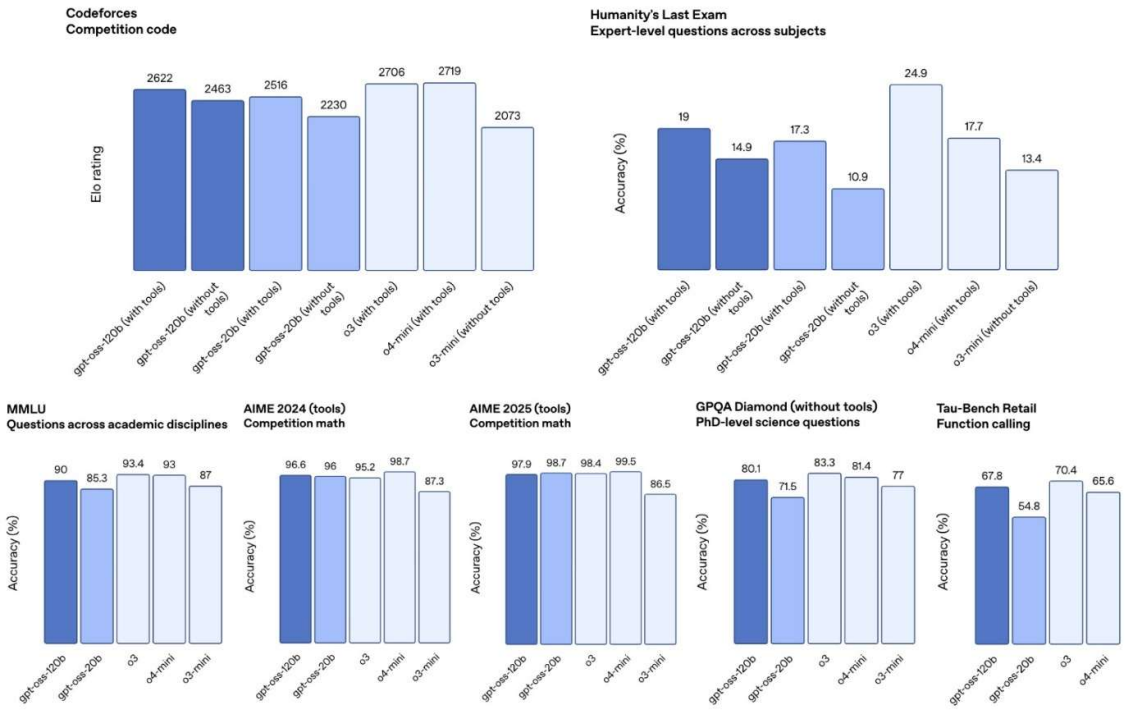
- **LayerNorm** uses **mean and variance** (centers data around zero).
- **RMSNorm** uses only **root mean square** (magnitude), so it's **simpler and faster**.
- RMSNorm **does not subtract the mean**, so it keeps the original direction of the vector.

MXFP4 Optimization

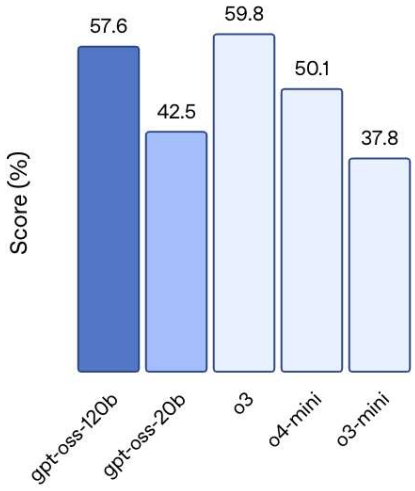
- Expert weights are stored in MXFP4 format:

- .blocks: FP4 values (4-bit floating point)
- .scales: int8 scale factors per block
- At runtime, they're decompressed into BF16 for actual computation.
- Benefit: The massive 120B model fits in a single high-memory GPU without sacrificing much quality.

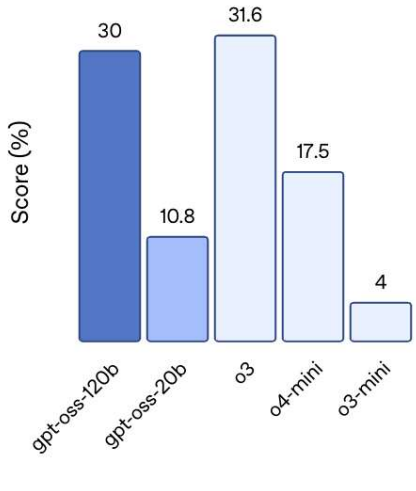
Benchmarks



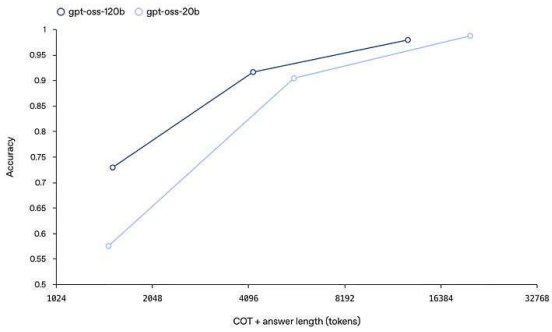
HealthBench
Realistic health conversations



HealthBench Hard
Challenging health conversations



AIME
Competition math



GPQA Diamond (tools)
PhD-level science questions