Lecture 20

- Position Encoding
- GPT-3.
- BERT.
- Other architectures

RoPE

- Modify $\vec{k}$, $\vec{q}$ based on their absolute position so that the score depends on the relative position.

$$\langle \vec{k}, \vec{q} \rangle = \|\vec{k}\|_2 \|\vec{q}\|_2 \cdot \cos\theta$$

Rotation matrix

$$R_t = \begin{bmatrix} \cos \omega_t & -\sin \omega_t \\ \sin \omega_t & \cos \omega_t \end{bmatrix}$$

$R_t \vec{q}$ → Rotates $\vec{q}$ by ω_t

$$\langle R_t \vec{q}_t , R_i \vec{k}_i \rangle = \vec{q}_t^{\,T} R_t^T R_i \vec{k}_i$$

R_t^T: rotation by $-\omega_t$; R_i: rotation by ω_i

$$M = \begin{bmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_{d/2} \end{bmatrix}$$

$$M_j = \begin{bmatrix} \cos(2\pi f_j \cdot t) & -\sin(2\pi f_j \cdot t) \\ \sin(2\pi f_j \cdot t) & \cos(2\pi f_j \cdot t) \end{bmatrix}$$

f_j's are fixed. Not learned.

Different "channels" will have different f_js.

So different timescales can be captured by different channels.

GPT-3 Architecture [Generative Pre-trained Transformer]

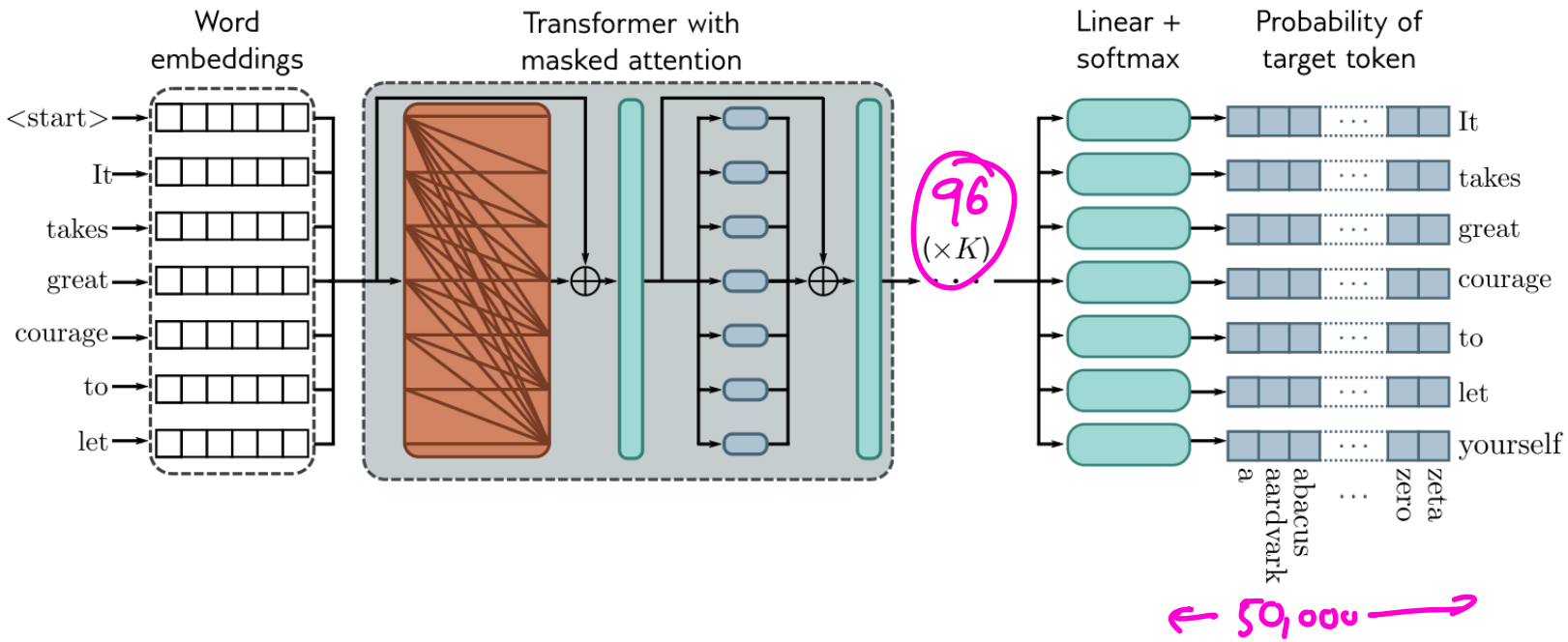


Figure 12.12 Training GPT3-type decoder network. The tokens are mapped to word embeddings with a special <start> token at the beginning of the sequence. The embeddings are passed through a series of transformer layers that use masked self-attention. Here, each position in the sentence can only attend to its own embedding and those of tokens earlier in the sequence (orange connections). The goal at each position is to maximize the probability of the following ground truth token in the sequence. In other words, at position one, we want to maximize the probability of the token It; at position two, we want to maximize the probability of the token takes; and so on. Masked self-attention ensures the system cannot cheat by looking at subsequent inputs. The autoregressive task has the advantage of making efficient use of the data since every word contributes a term to the loss function. However, it only exploits the left context of each word.

"Decoder-only" models.

- Focus: generate the next token in a sequence.
- Masked / causal self attention training.

→ Only attend to current and previous tokens.

- Cross-entropy loss training
- Inference

→ Sample token from output probabilities.

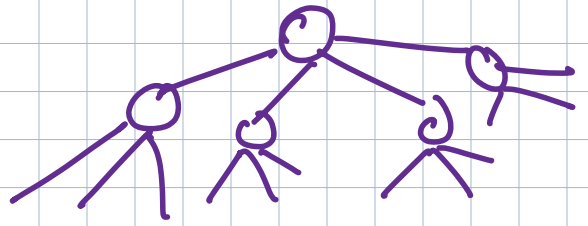
→ Auto-regressive generation : feedback generated token as input.

* Can reuse much of the computations for attention * important for compute optimization.

→ Sampling strategies:

(1) Beam search

→ Truncated best-first search.



(2) Top-k sampling.

Size: 96 transformer layers

96 heads in self-attention layers

Key-Value-Query dimension 128.

Embedding 12288 dimensions (!).

Input sequence 2048 tokens long.

Total: 175 billion parameters.

Trained with. 300 billion tokens.

BERT – Encoder only model.

Bidirectional Encoder Representations from Transformers.

Goal: Learn general information about the statistics of Language

Use fine-tuning to adapt to solve specific task.

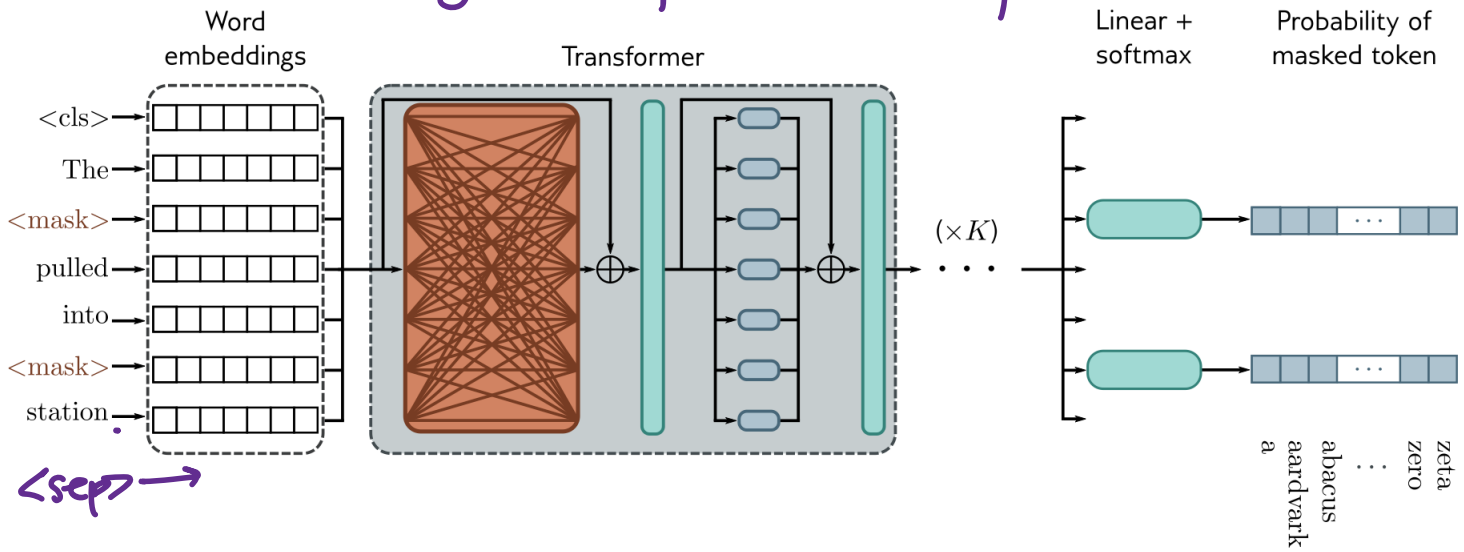


Figure 12.10 Pre-training for BERT-like encoder. The input tokens (and a special <cls> token denoting the start of the sequence) are converted to word embeddings. Here, these are represented as rows rather than columns, so the box labeled "word embeddings" is $\mathbf{X}^T$. These embeddings are passed through a series of transformer layers (orange connections indicate that every token attends to every other token in these layers) to create a set of output embeddings. A small fraction of the input tokens are randomly replaced with a generic <mask> token. In pre-training, the goal is to predict the missing word from the associated output embedding. To this end, the outputs corresponding to the masked tokens are passed through softmax functions, and a multiclass classification loss (section 5.24) is applied to each. This task has the advantage that it uses both the left and right context to predict the missing word but has the disadvantage that it does not make efficient use of data; here, seven tokens need to be processed to add two terms to the loss function.

Pre-training : Self-supervision.

→ Masked token prediction

→ Small fraction of them are masked.

→ <Cls> start token

→ <sep> token to separate sentences.

→ Cross-entropy loss to predict masked tokens

* Sentence adjacency prediction.

Sent A Sent B.

Predict if Sent B immediately follows A.

→ Predict this on the sep token.

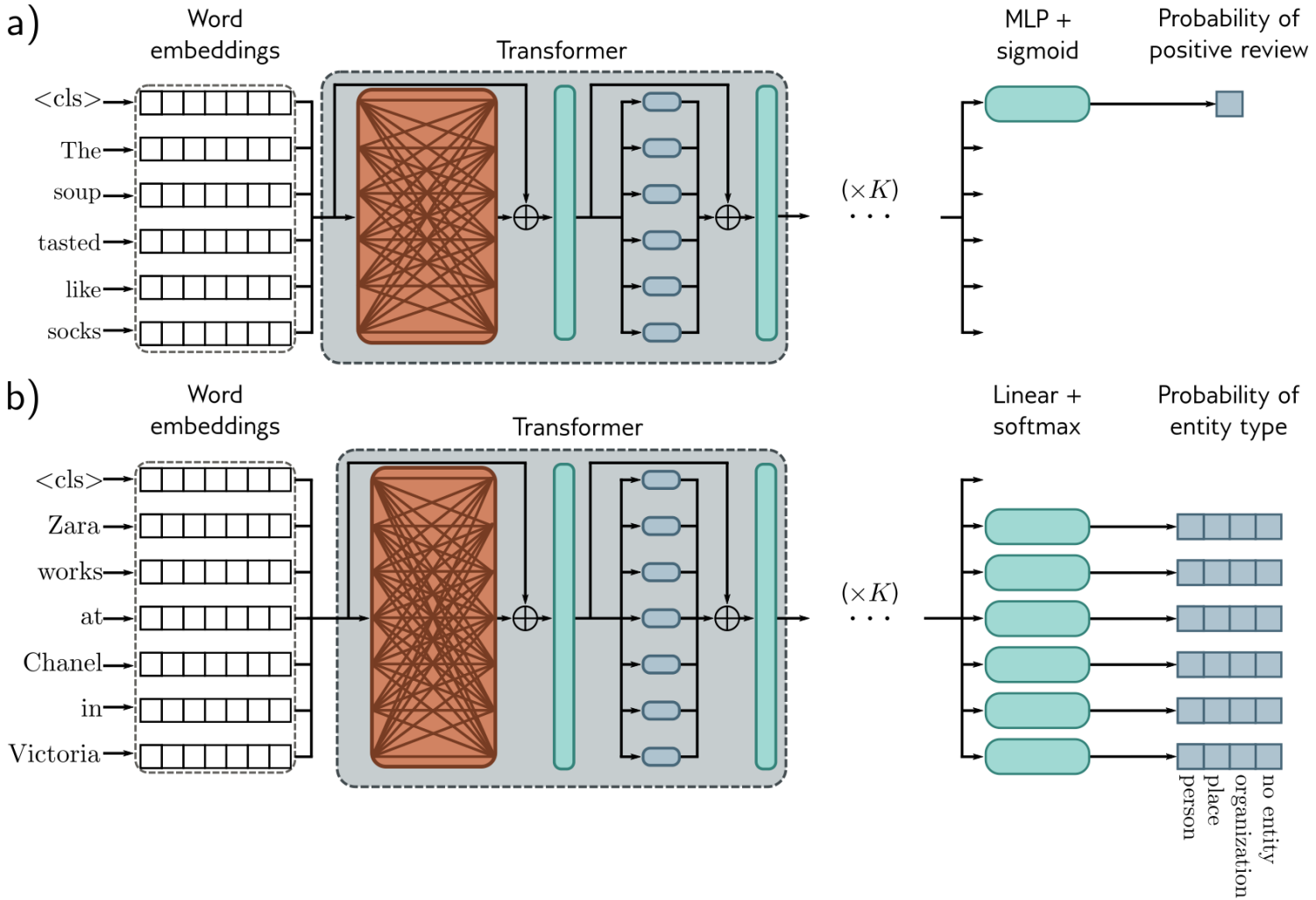


Figure 12.11 After pre-training, the encoder is fine-tuned using manually labeled data to solve a particular task. Usually, a linear transformation or a multi-layer perceptron (MLP) is appended to the encoder to produce whatever output is required. a) Example text classification task. In this sentiment classification task, the <cls> token embedding is used to predict the probability that the review is positive. b) Example word classification task. In this named entity recognition problem, the embedding for each word is used to predict whether the word corresponds to a person, place, or organization, or is not an entity.

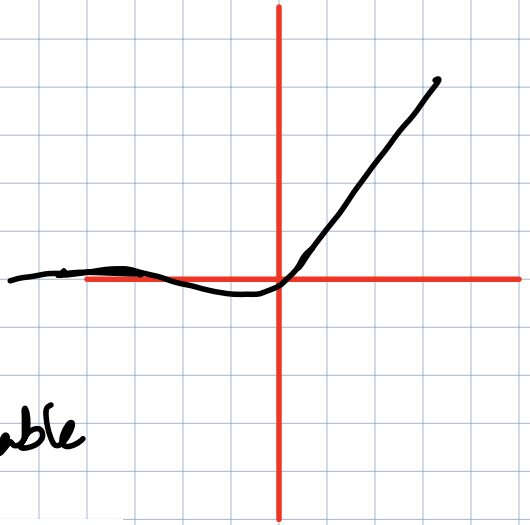
Gated Linear Units.

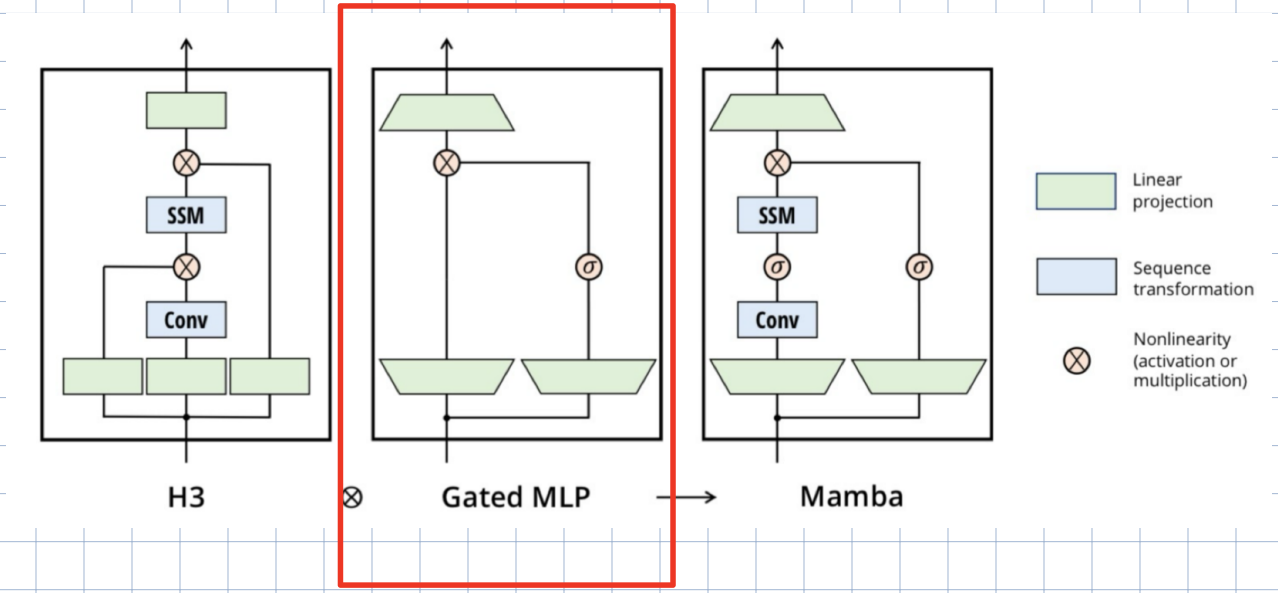
$$GLU(x) = (Wx+b) * \text{sigmoid}(Vx+c)$$

$$SwiGLU(x) = (Wx+b) * \text{Swish}(Vx+c)$$

$$\text{swish}(x) = x \cdot \text{sigmoid}(\beta \cdot x).$$

learnable





Mixture of experts models

Key idea: MLPs use a lot of parameters at inference time. ↑compute.

Can we use fewer through specialization?

Switch transformer

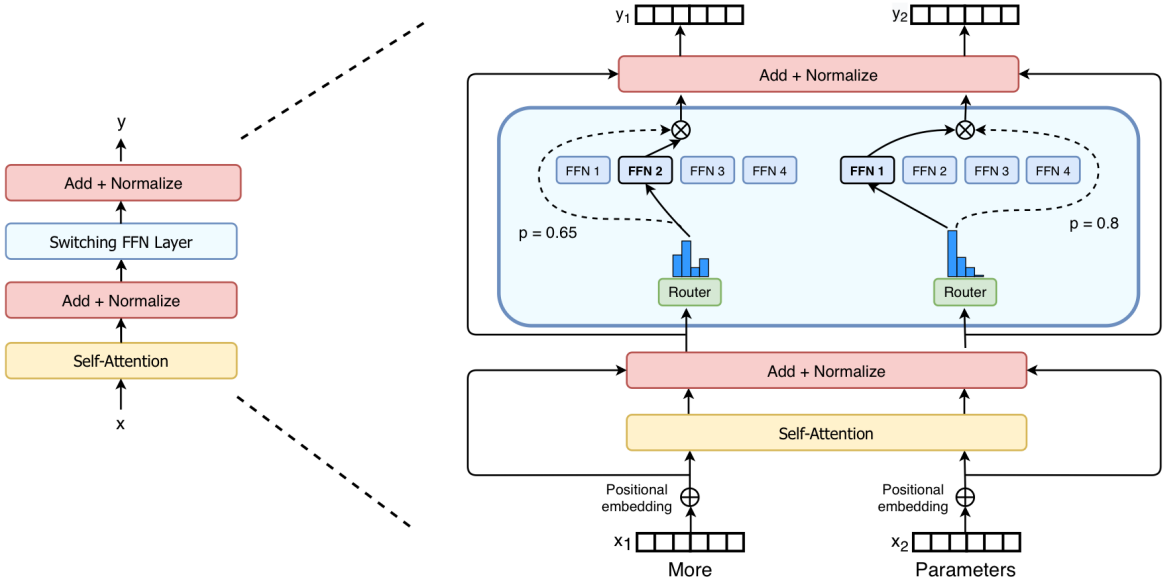


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

Fedus et al 2021.

OLMoE.

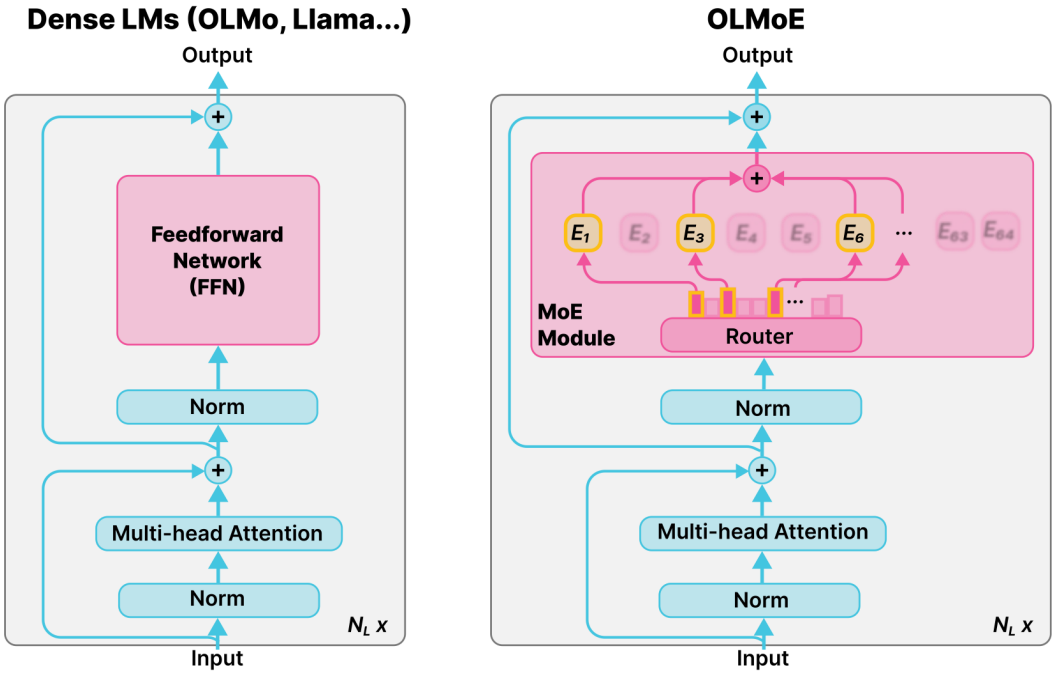
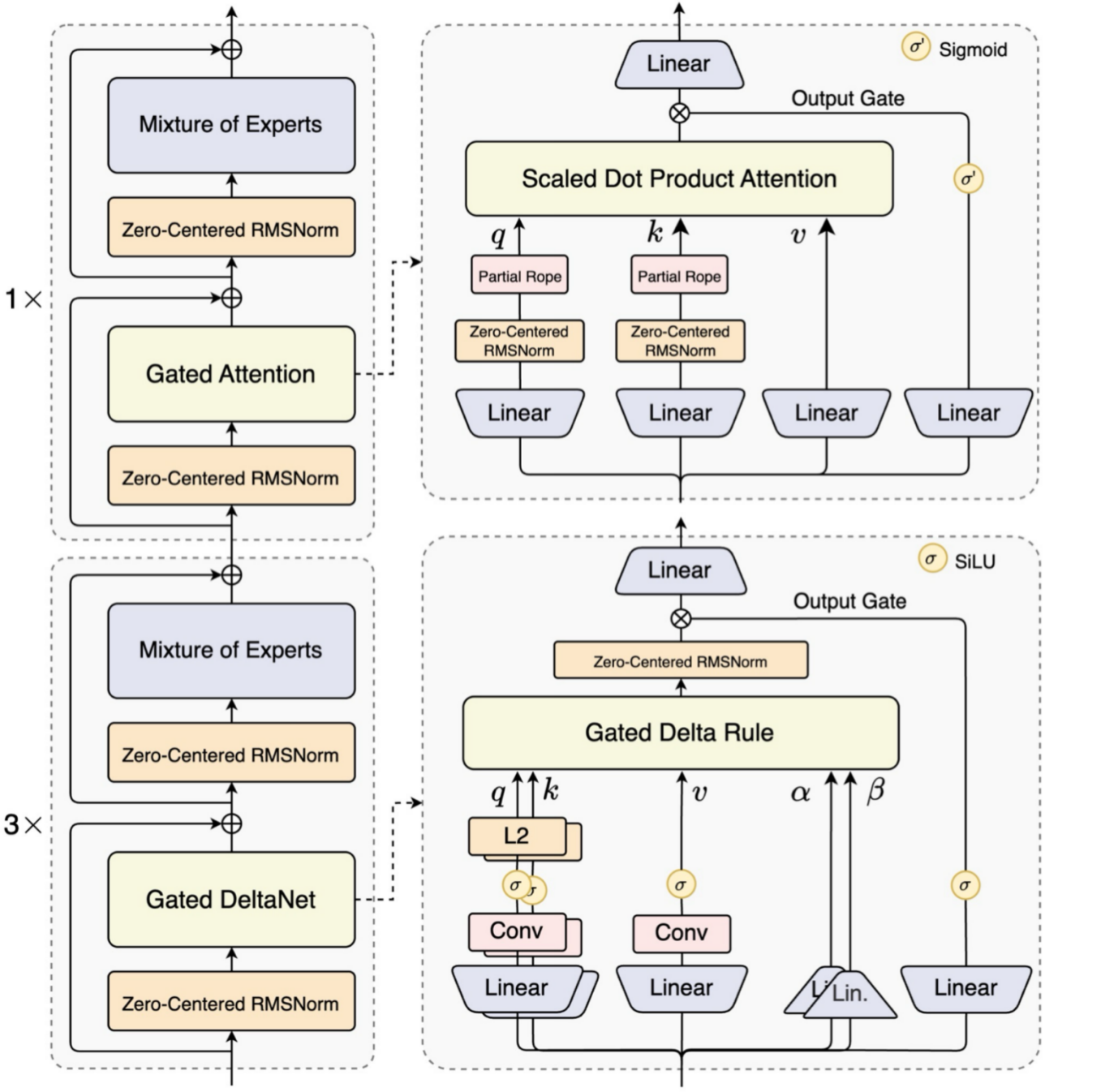


Figure 2: **Comparison of the architecture of dense LMs and MoE models like OLMoE.** The figure excludes some details, e.g., **OLMoE-1B-7B** also uses QK-Norm (§4.2.5).

Design choice	Description	Experiment	OLMoE-1B-7B
Active params	# active parameters per input token	§4.1.1	1.3B active
Total params	Total # of parameters in the model	§4.1.1	6.9B total
Expert granularity	Using fine-grained small experts vs. a few large experts [39]	§4.1.2	64 small experts with 8 activated
Expert sharing	Whether or not to include a shared expert [39]	§4.1.3	No shared expert
Routing algorithm	How inputs are assigned to experts, e.g., assignment on a per token basis (e.g., 2 experts per token) or per expert basis (e.g., 2 tokens per expert), and whether or not all tokens get assigned or some get dropped [58, 219]	§4.1.4	Dropless [58] MoE with token choice
Sparse upcycling	Whether to start from a dense model [85, 211]	§4.1.5	Not used
Load balancing loss	Auxiliary loss to penalize unequal assignment to experts that may harm performance [154]	§4.1.6	Used with weight 0.01
Router z-loss	Auxiliary loss to penalize large logits in the router that may cause instabilities [221]	§4.1.7	Used with weight 0.001

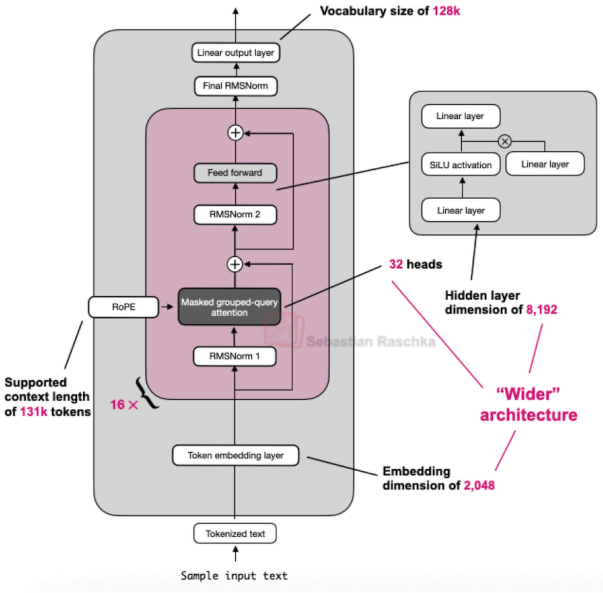
Table 1: **Key MoE design choices and our setup for OLMoE-1B-7B based on our experiments.** Full configuration for **OLMoE-1B-7B** is in Appendix B.

Qwen next

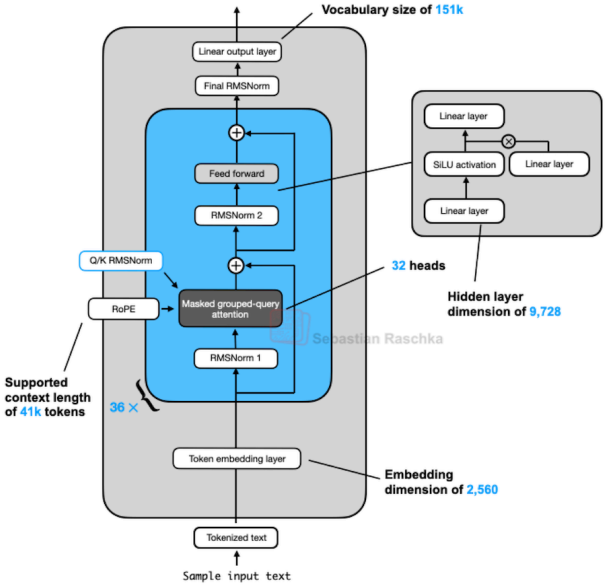

1×
Mixture of Experts
Zero-Centered RMSNorm
Gated Attention
Zero-Centered RMSNorm
3×
Mixture of Experts
Zero-Centered RMSNorm
Gated DeltaNet
Zero-Centered RMSNorm
Linear
Output Gate
Sigmoid
Scaled Dot Product Attention
q
k
v
Partial Rope
Partial Rope
Zero-Centered RMSNorm
Zero-Centered RMSNorm
Linear
Linear
Linear
Linear
Linear
Output Gate
SiLU
Zero-Centered RMSNorm
Gated Delta Rule
q
k
v
α
β
L2
Conv
Conv
Linear
Linear
Lin.
Linear

Sebastian Rashka blog.

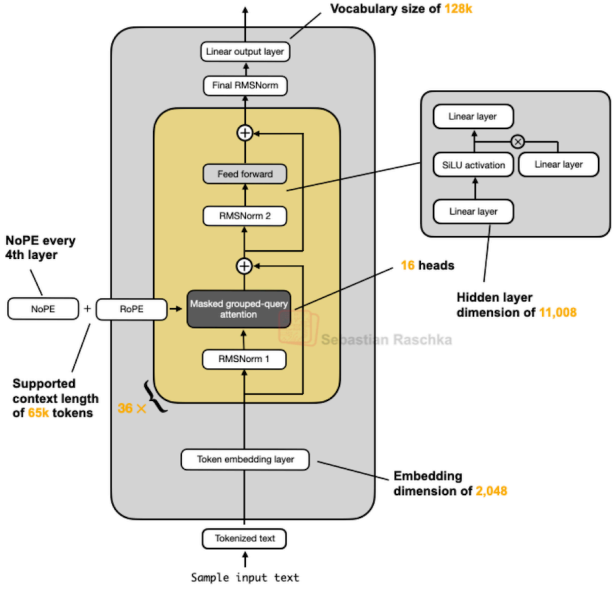
Llama 3.2 1B



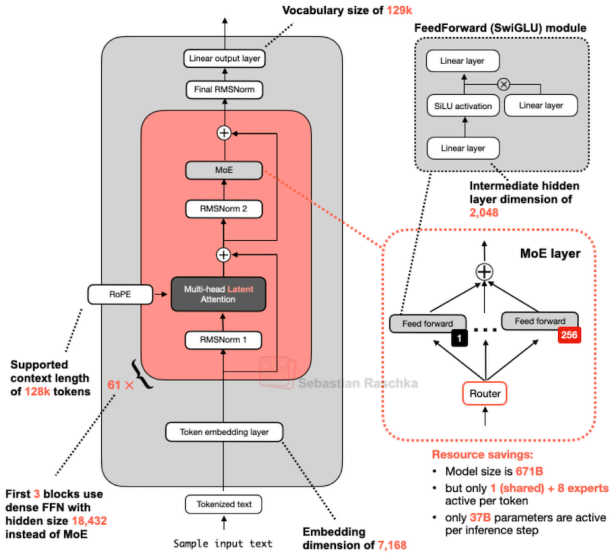
Qwen3 4B



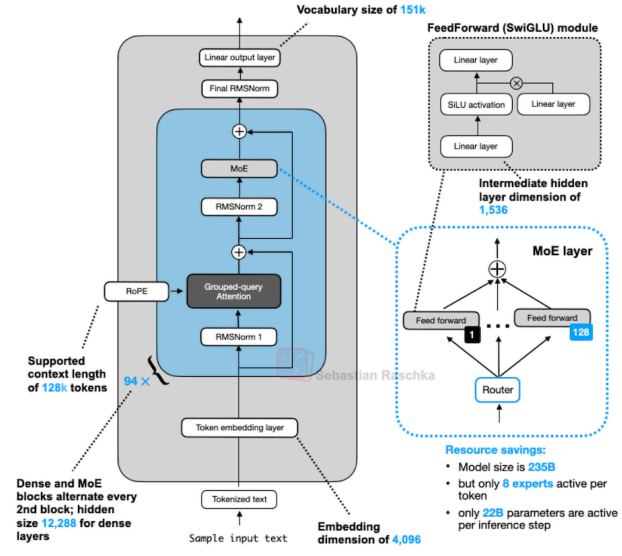
SmolLM3 3B



DeepSeek V3 (671B)



Qwen3 235B-A22B



Kimi K2 (1 trillion)