Project Machine Learning Systems
02 Transformer Architecture

Philipp Ortner
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

PUBLIC DOMAIN

Last update: Oct 28, 2026

Announcements / Org

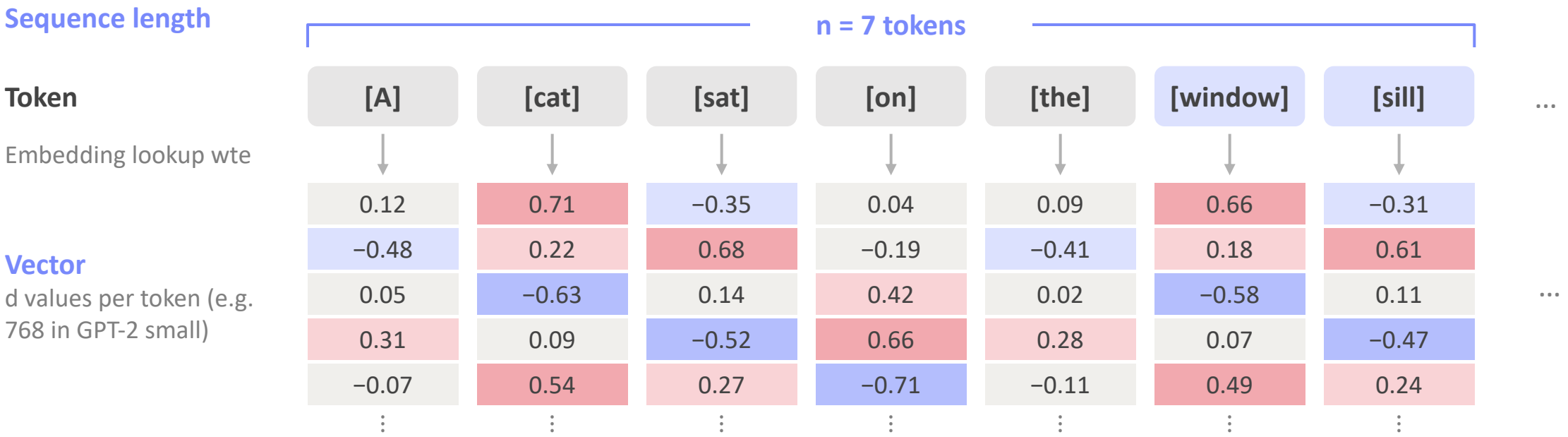
- **#1 Hybrid for session 01 and 02**
  - Hybrid lectures (in-person, zoom): https://tu-berlin.zoom-x.de/j/4288401545?pwd=Z044YXd6MXUrUUlUT1RuZXh3TEg5Zz09
  - Website: https://philipportner.github.io/pmls.html
- **#2 In-person for session 03, 04, and 05**
  - Larger room (FR-713)
  - Interactive discussion rounds
- **#3 A1 published**
  - You should have access on git.tu-berlin.de
- **#4 No sign-up in Moses needed**
  - Tag of A1 -> grade

Agenda

- **Recap**
- **LLM Walkthrough**
  - Transformer Architecture
  - Decoding and generation
- **Roofline**
- **Prefill and Decode**
- **KV Caching**
- **Mixture of Experts**
- **Project DAMSIE**

From Text to Tokens

- **Subword tokens: reuse word roots, learned from data**
- **Cost: extra training step; efficiency depends on the corpus**
- **Each token is mapped to a vector**



The GPT-2 Tokenizer

Vocabulary

- GPT-2: **50,257 tokens** = 256 bytes + 50,000 merges + <|endoftext|>
- Modern LLMs: **100k–200k** tokens, covering more languages
- **vocab.json** maps tokens to IDs; embedding table **wte**[50257, 768]

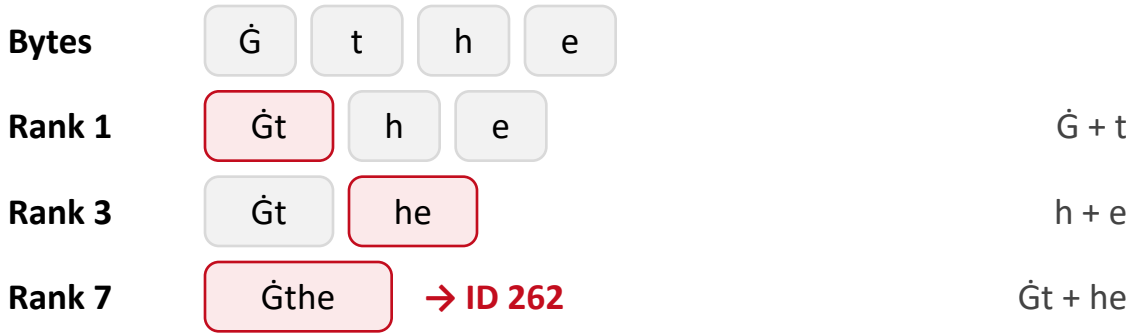
Special tokens

- **[BOS]**, **[EOS]**: start and end of a sequence
- **[PAD]**: fills short sequences; attention mask is 0
- GPT-2: <|endoftext|> (ID 50256) serves as all three

Byte Pair Encoding (BPE)

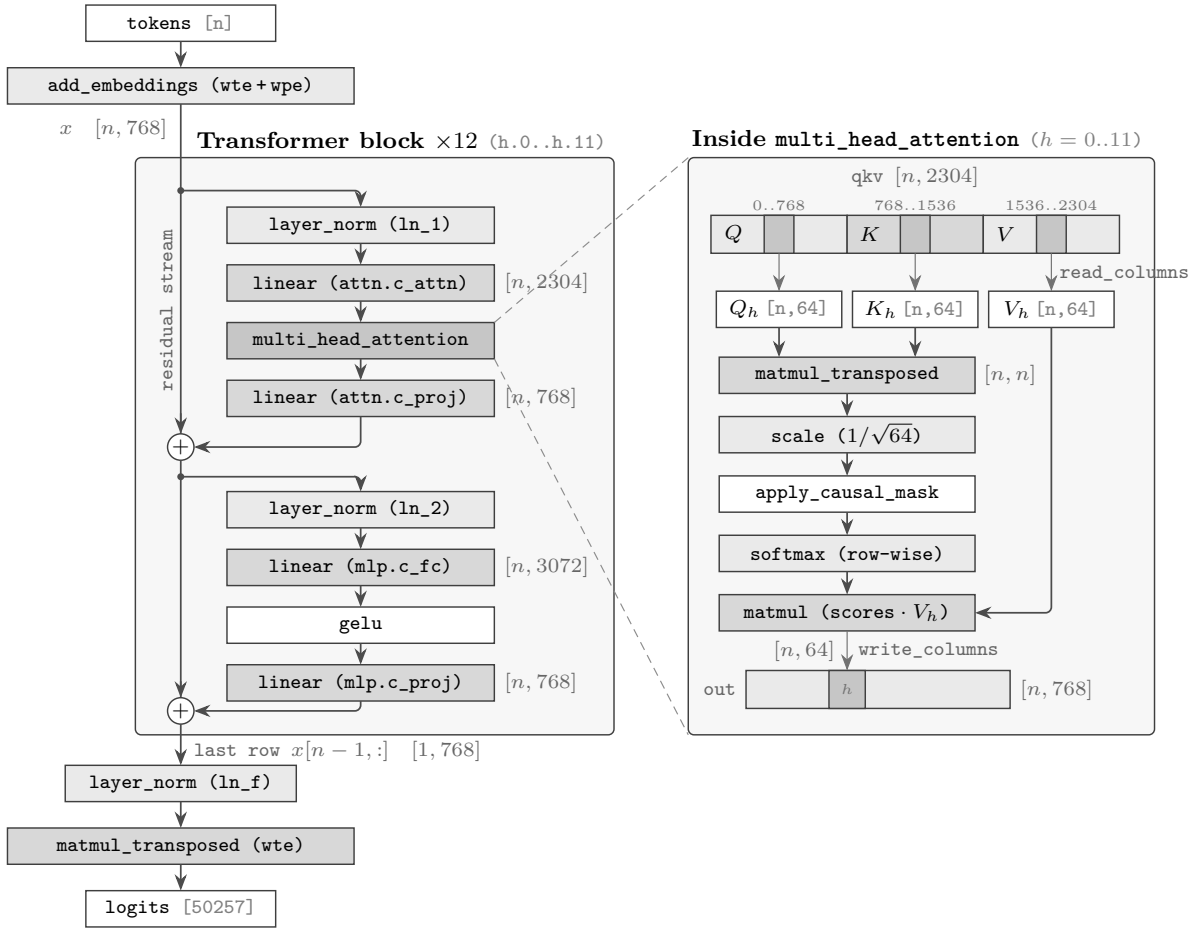
- Regex pre-tokenization splits text into pieces before merging
- 50,000 merges learned once, offline; stored in **merges.txt**
- Inference: start from bytes, repeatedly merge the adjacent pair with the smallest rank

Example: " the" (Ġ = space byte)



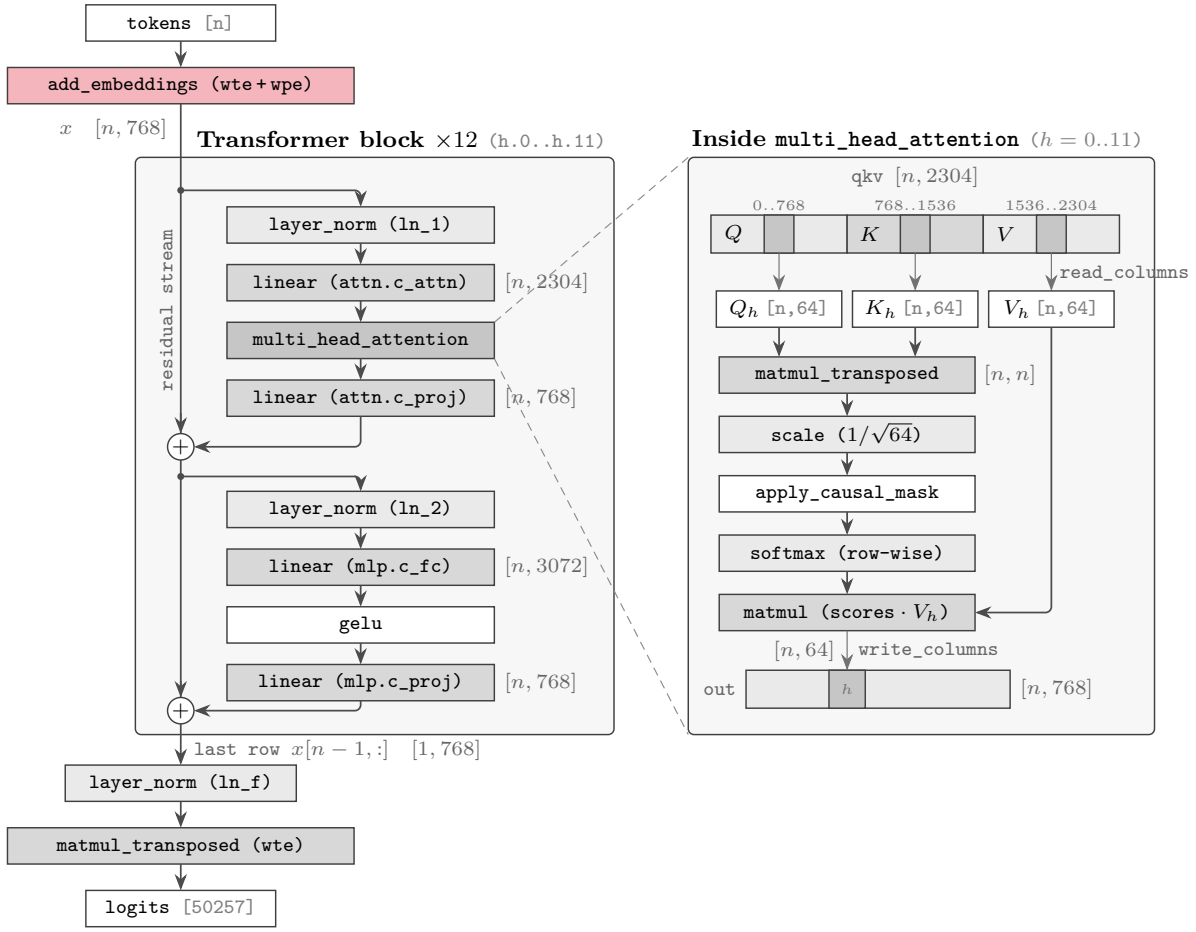
Source: R. Sennrich, B. Haddow, A. Birch: Neural Machine Translation of Rare Words with Subword Units. ACL 2016.

Walkthrough



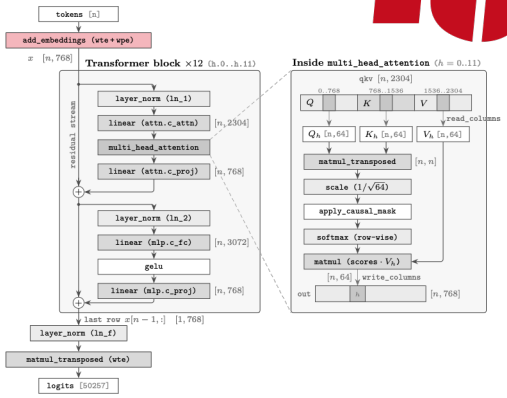
A. Vaswani et al.: Attention Is All You Need. NeurIPS 2017.

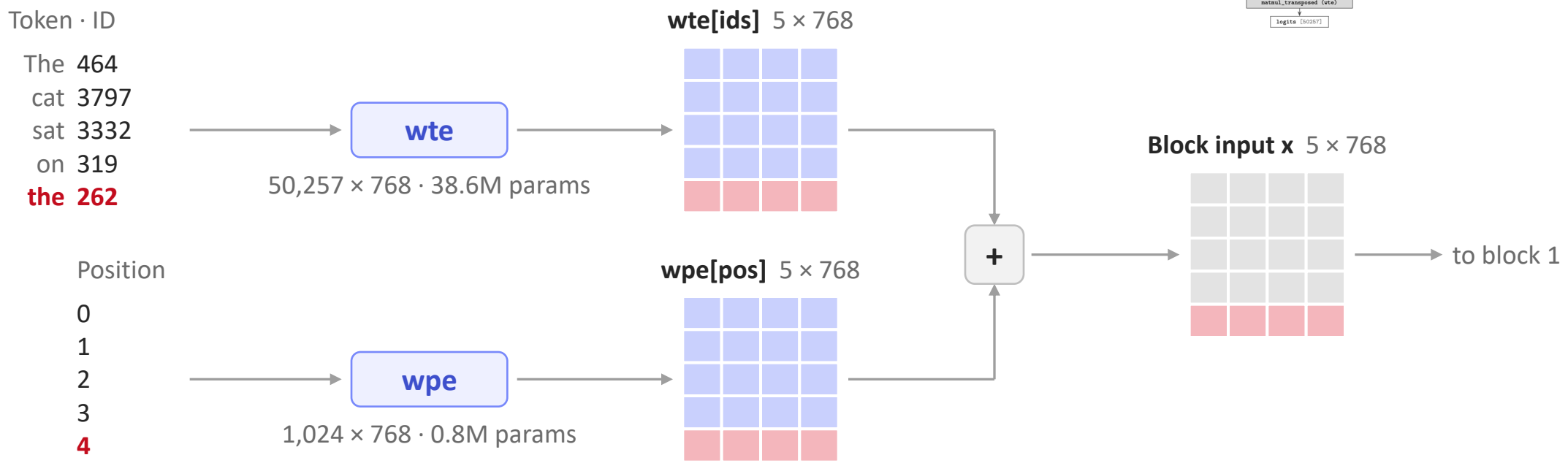
Token and Positional Embeddings



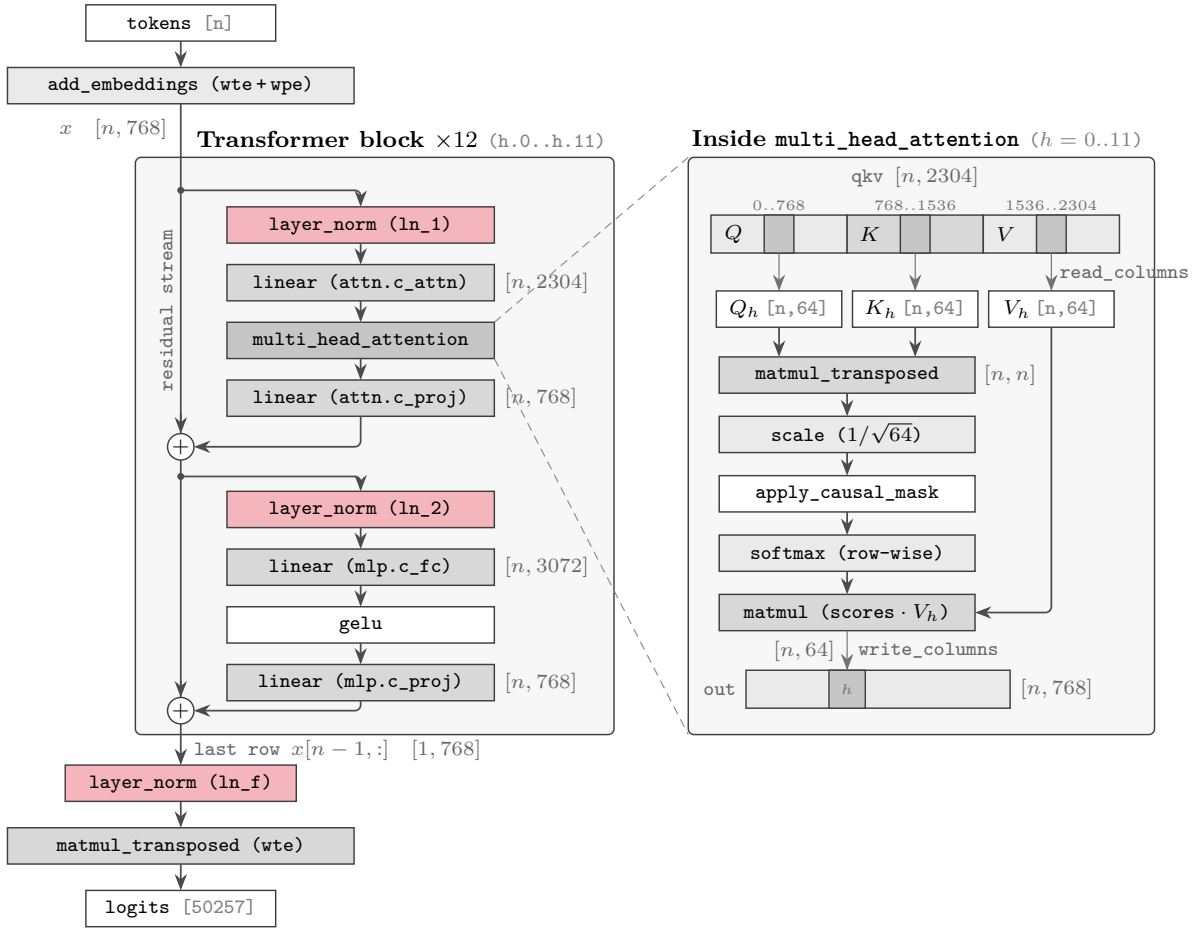
Token and Positional Embeddings

- **wte (token embedding table): 50,257 × 768 = 38.6M parameters**
- **wpe (learned position embedding table): 1,024 × 768 = 0.8M parameters**
- **Block input x = wte[ids] + wpe[pos]: T IDs → (T, 768)**



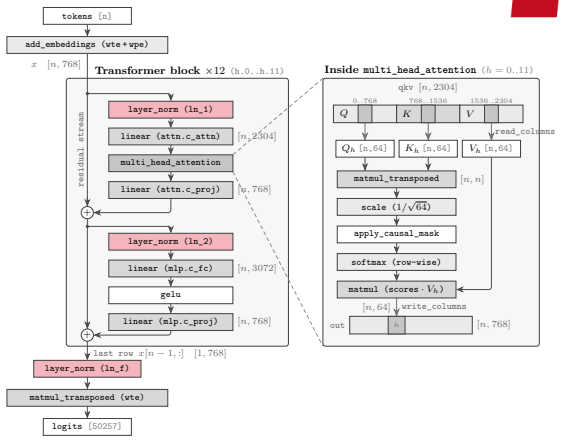


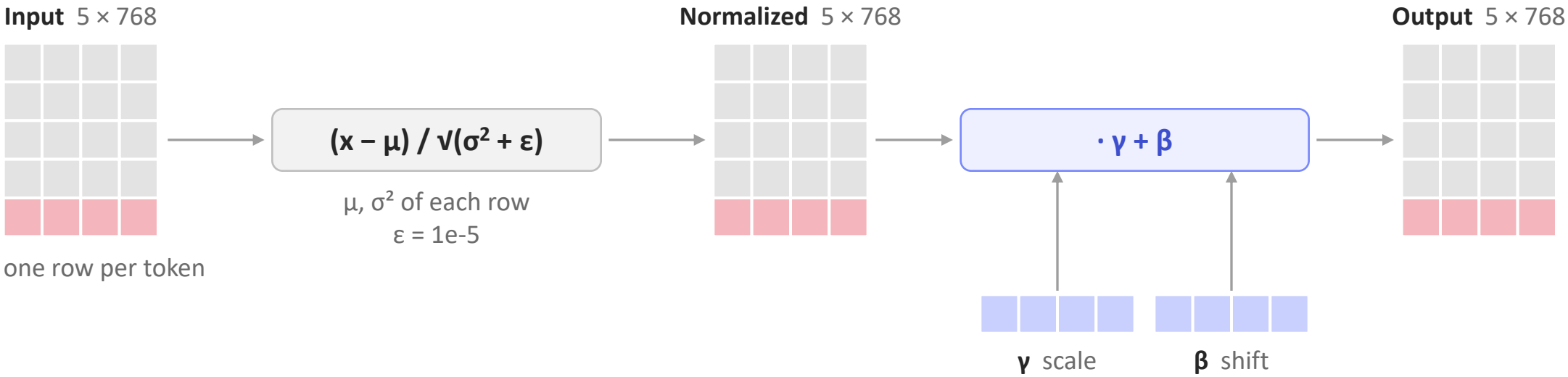
Layer Normalization



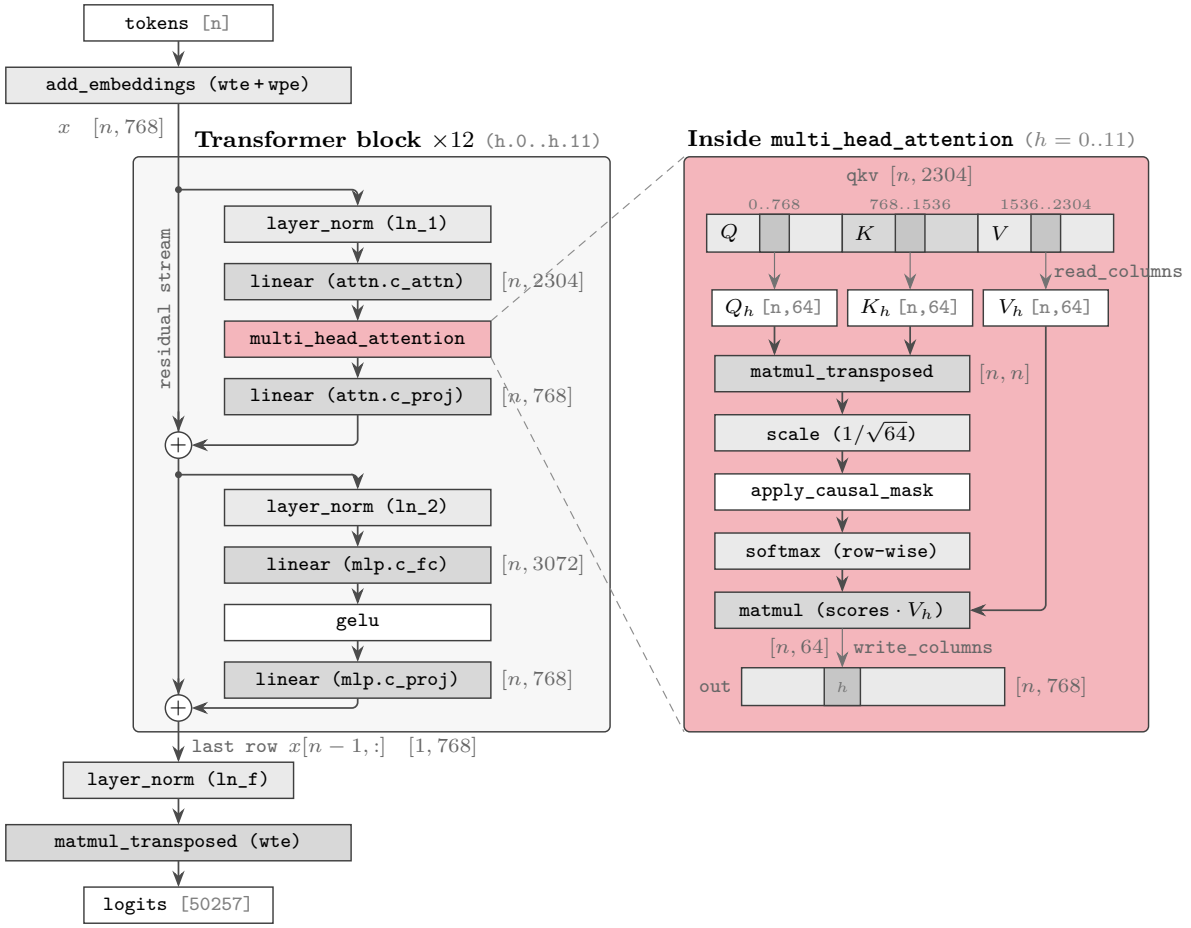
Layer Normalization

- **Normalize each 768-dim token vector to mean 0, variance 1 (ε = 1e-5)**
- **Learned scale γ (weight) and shift β (bias), 768 values each**
- **ln_1 before attention, ln_2 before the MLP, ln_f after the last block**
- **25 LayerNorms: 2 per block × 12 blocks + 1 final**

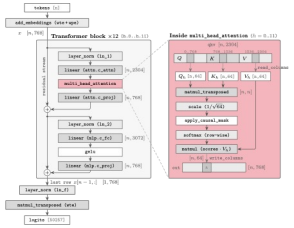




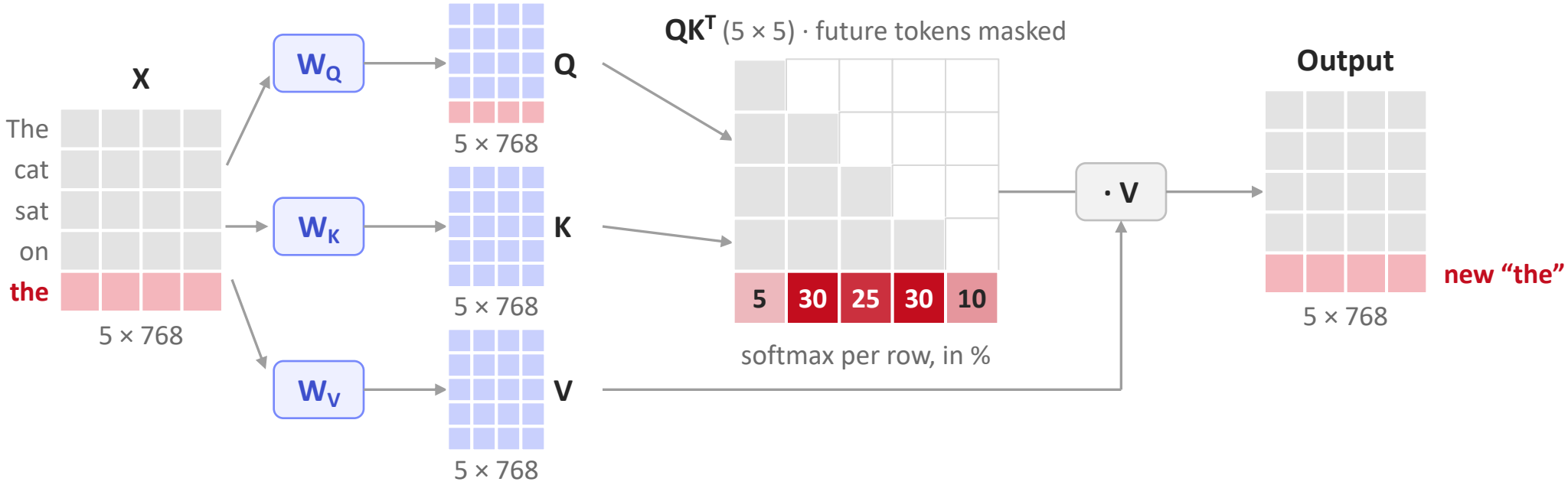
The Transformer Block



Masked Causal Self-Attention



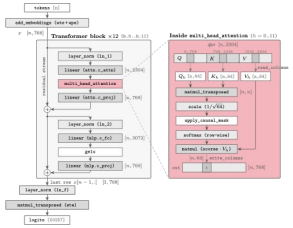
- **Stacking all tokens into X computes every token's output with a few matrix products**
- **Causal mask: future scores become −∞ before softmax**



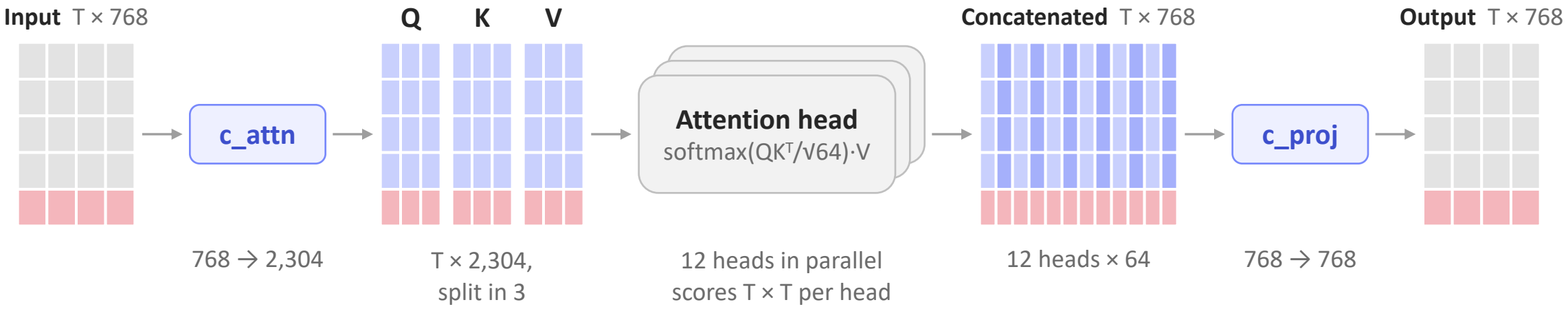
$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d}) \cdot V$$

Source: A. Vaswani et al.: Attention Is All You Need. NeurIPS 2017.

Multi-Head Self-Attention



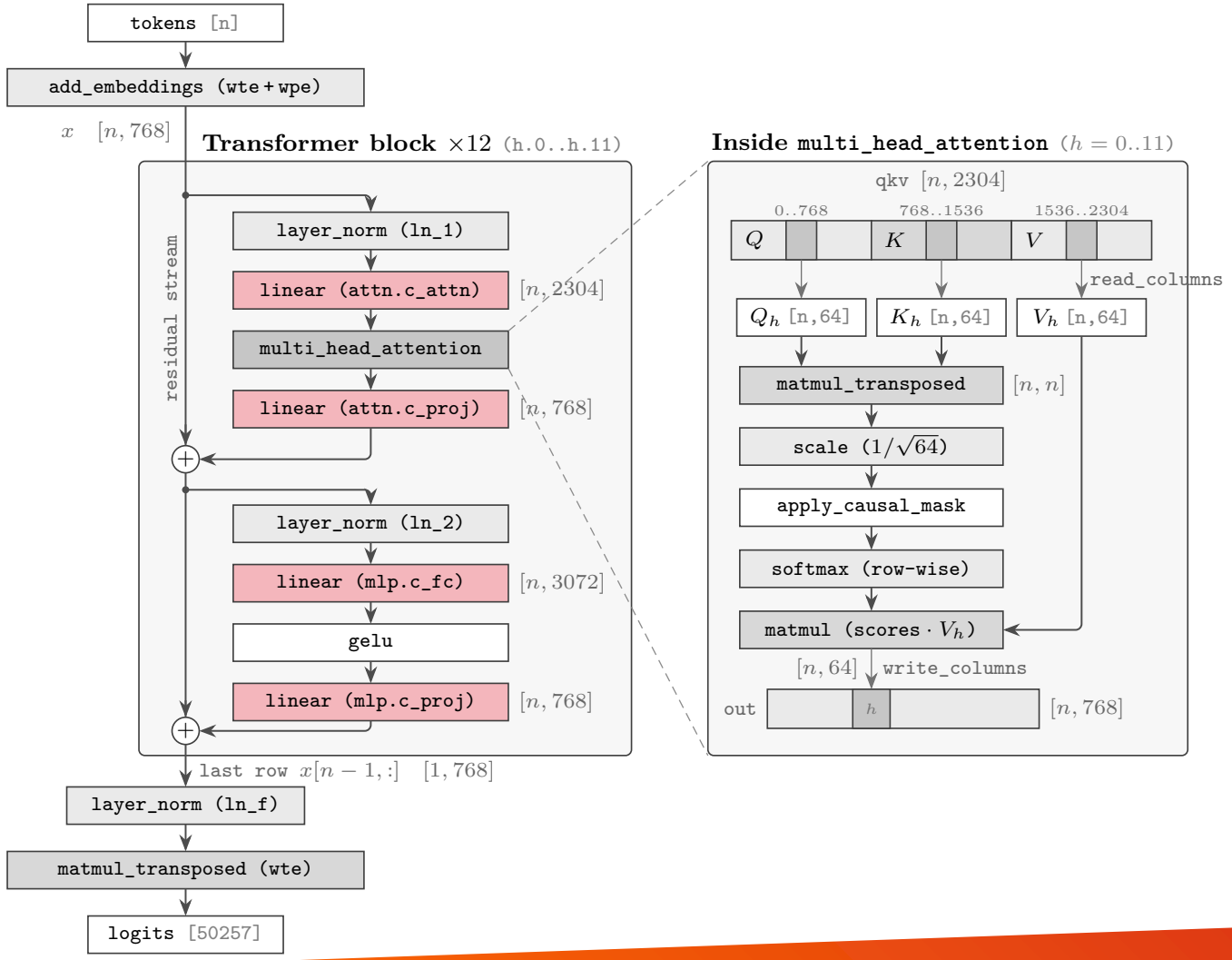
- **12 heads attend in parallel, each on its own 64-dim slice of Q, K and V**



(T, 768) → (T, 2,304) → (12, T, 64) → (12, T, T) → (T, 768)

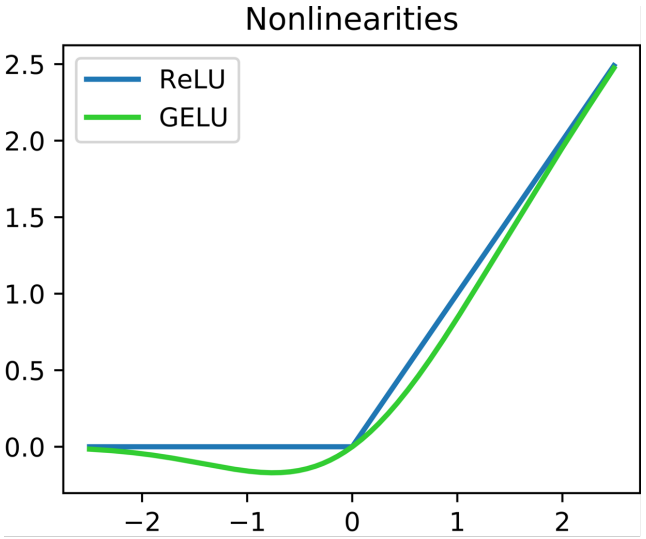
T = sequence length; GPT-2 has 12 heads of 64 dims each.

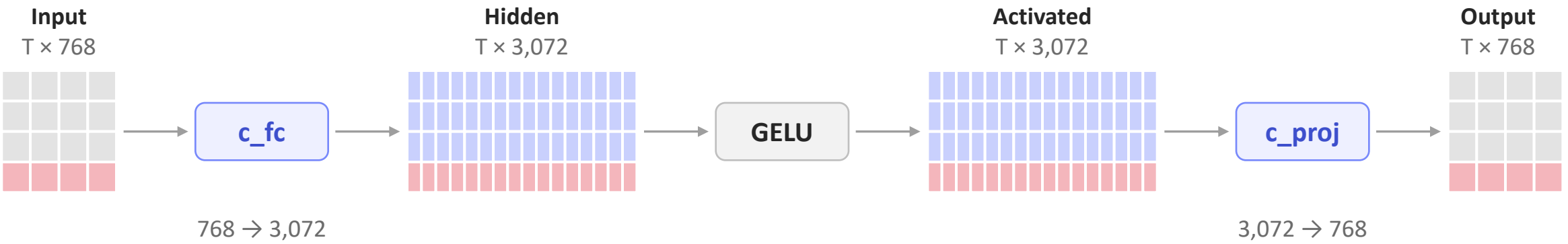
Feed-Forward Network



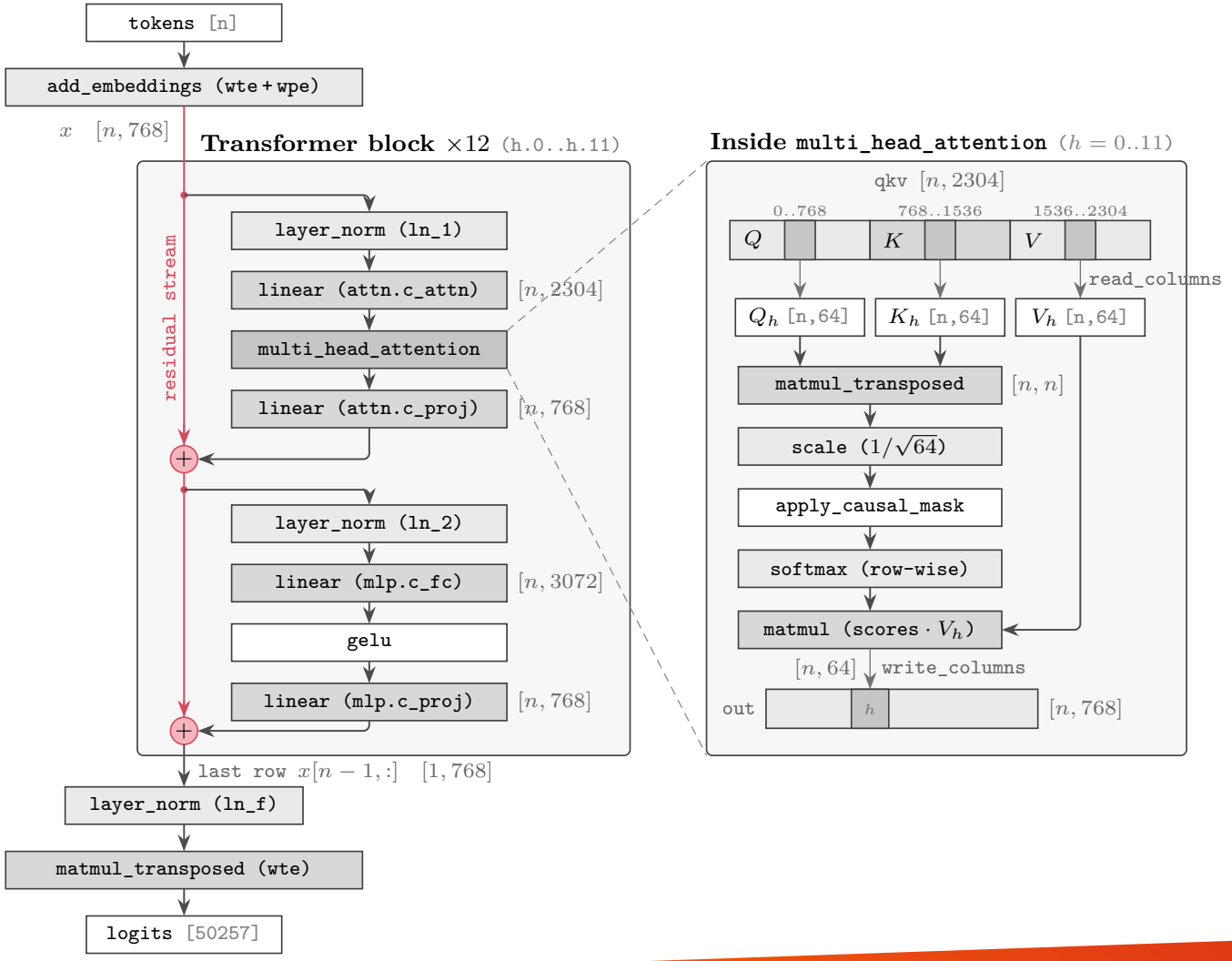
Feed-Forward Network

- **mlp.c_fc (expansion layer): 768 → 3,072 (4×)**
- **GELU activation (tanh approximation)**
- **mlp.c_proj (projection back): 3,072 → 768**
- **Per token: (T, 768) → (T, 3,072) → (T, 768)**
- **4.7M parameters per block, two thirds of the block**



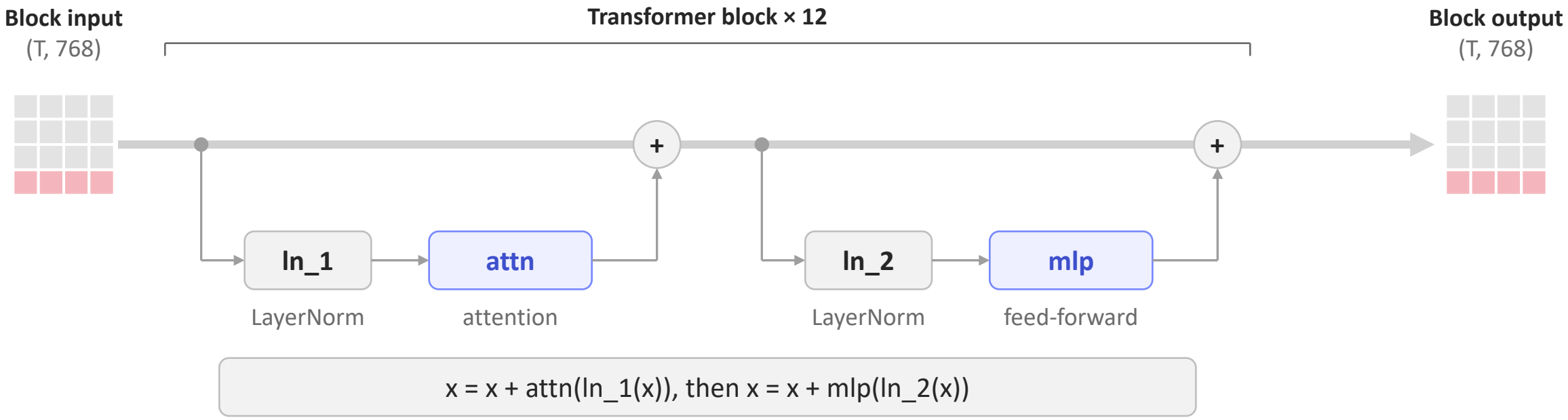


Residual Connections

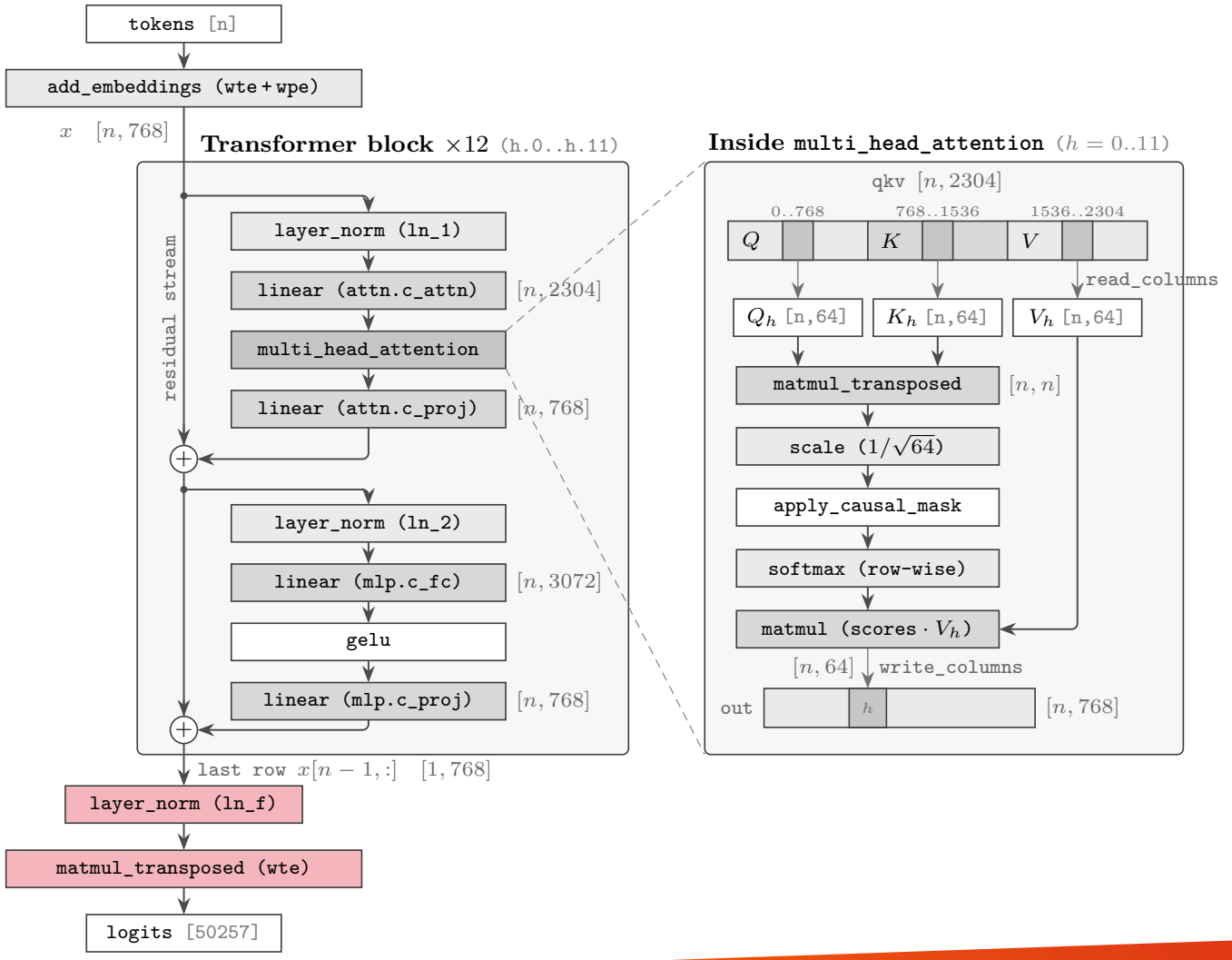


Residual Connections

- **Residual stream of shape (T, 768) runs through all 12 blocks**
- **Each sublayer adds to it: x = x + f(LN(x))**

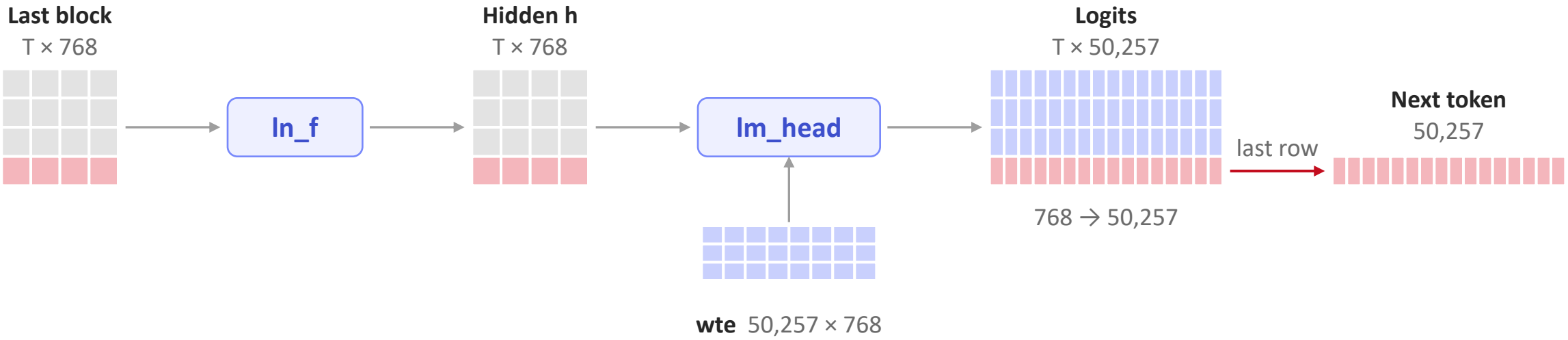


Output Head

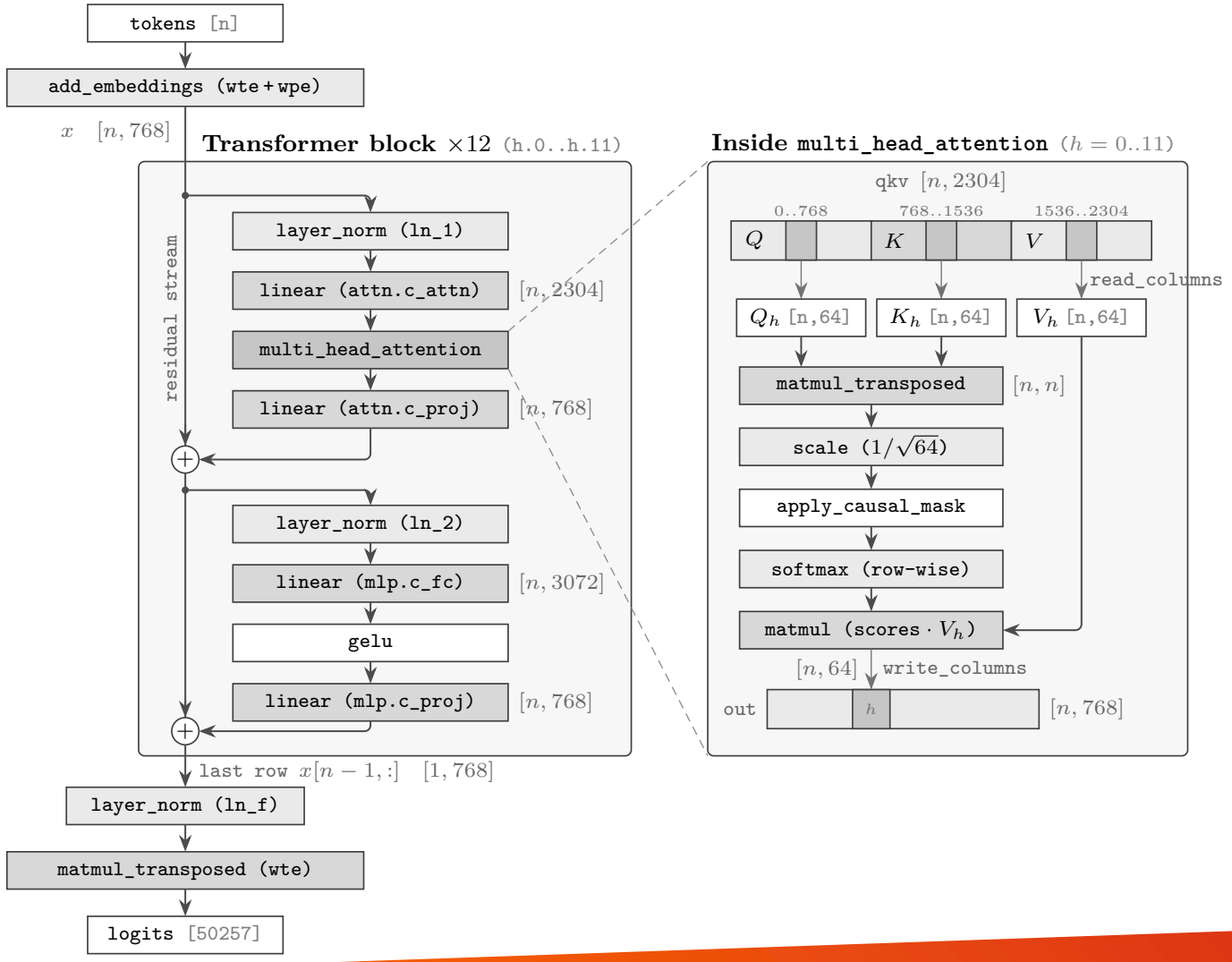

tokens [n]
add_embeddings (wte + wpe)
x [n, 768]
Transformer block ×12 (h.0..h.11)
residual stream
layer_norm (ln_1)
linear (attn.c_attn)
[n, 2304]
multi_head_attention
linear (attn.c_proj)
[n, 768]
layer_norm (ln_2)
linear (mlp.c_fc)
[n, 3072]
gelu
linear (mlp.c_proj)
[n, 768]
last row x[n − 1, :] [1, 768]
layer_norm (ln_f)
matmul_transposed (wte)
logits [50257]
Inside multi_head_attention (h = 0..11)
qkv [n, 2304]
0..768
768..1536
1536..2304
Q
K
V
read_columns
Q_h [n,64]
K_h [n,64]
V_h [n,64]
matmul_transposed
[n, n]
scale (1/√64)
apply_causal_mask
softmax (row-wise)
matmul (scores · V_h)
[n, 64]
write_columns
out
h
[n, 768]

Output Head

- ln_f (final LayerNorm) on the last block's output, (T, 768)
- lm_head (output projection): 768 → 50,257
- Logits = h · wte^T, shape (T, 50,257)
- Generation uses only the last position: (50,257)



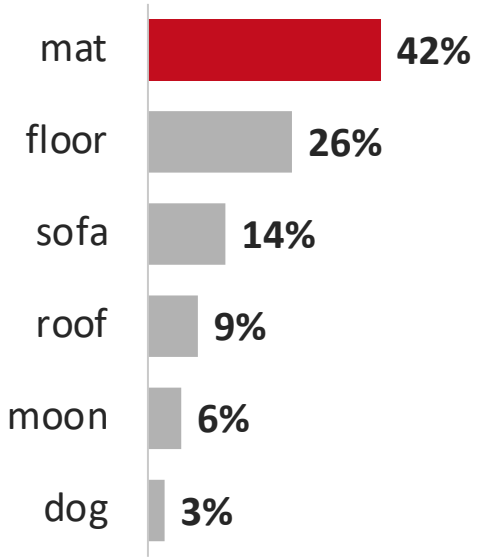
Walkthrough



From Logits to the Next Token

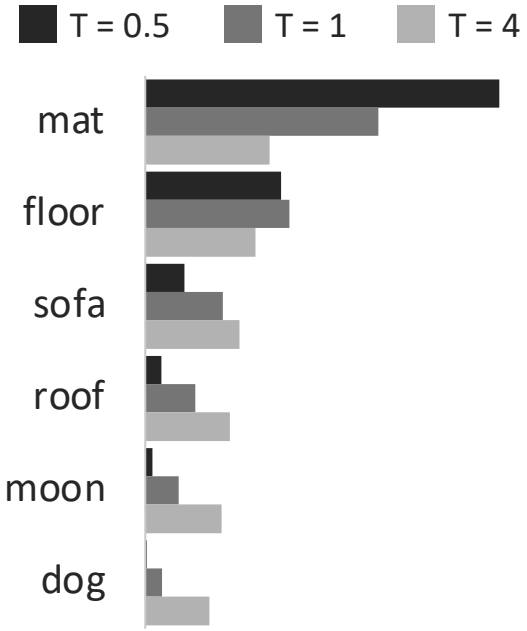
Greedy

highest probability wins



Always picks mat
no randomness

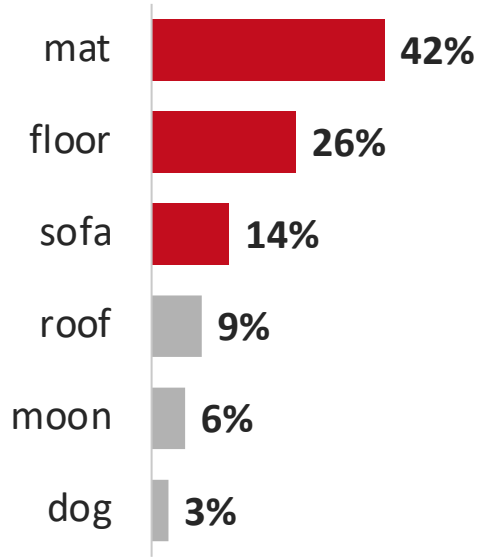
Temperature



mat: 64%, 42%, 22%
sharper → flatter

Top-k (k = 3)

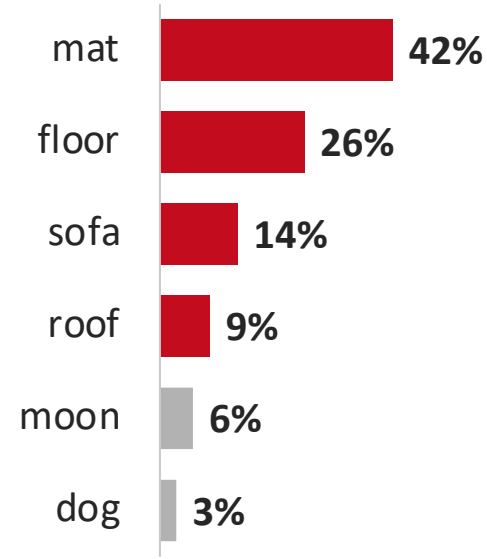
keep the 3 highest



Samples among 3 tokens
renormalized: 51/32/17%

Top-p (p = 0.9)

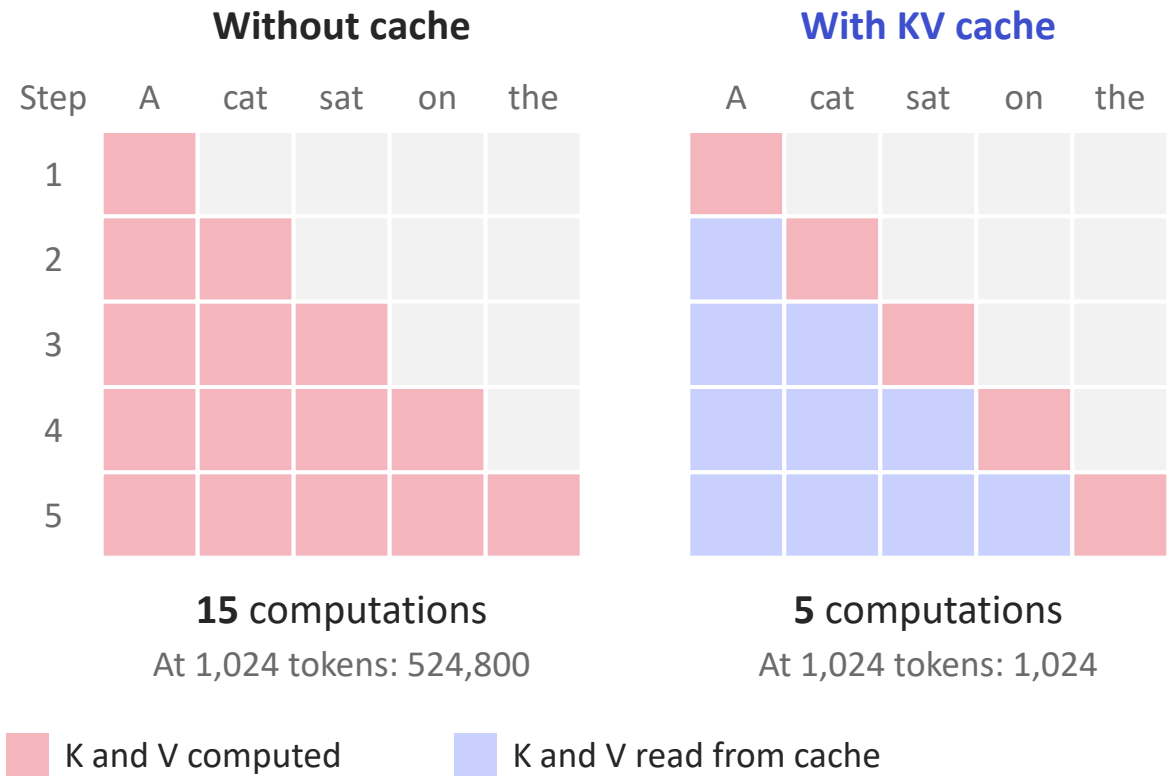
smallest set reaching 90%



Samples among 4 tokens
renormalized: 46/29/15/10%

KV Caching

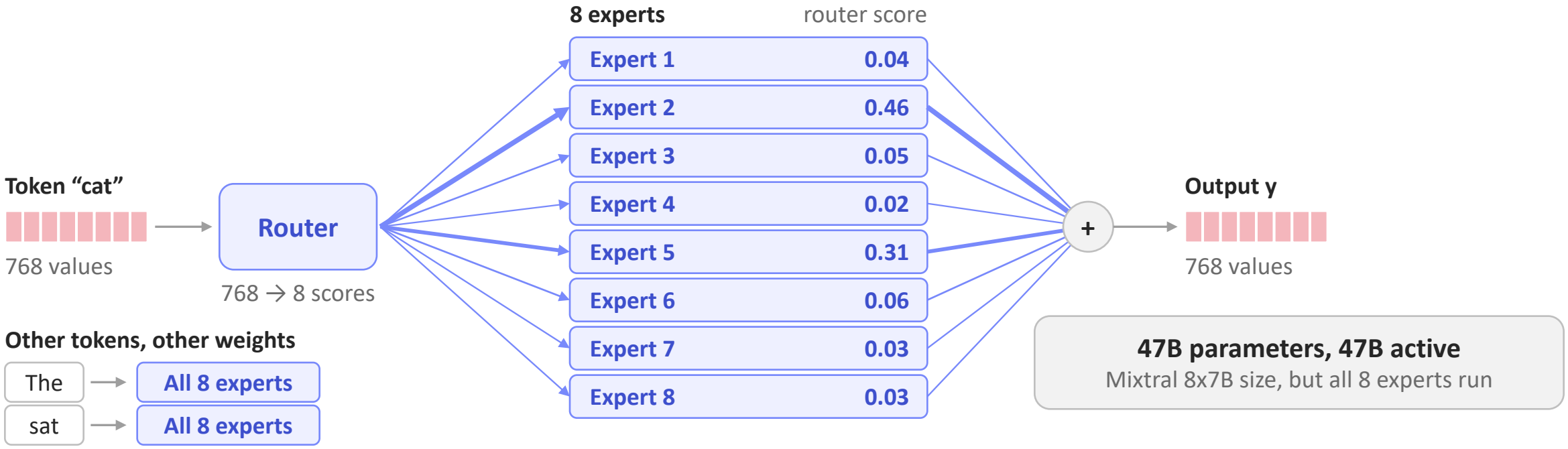
- **Without cache: every step recomputes K and V for all previous tokens**
- **Cache K and V per layer; each new step computes only the new token**
- **Cost per step: $O(n^2) \rightarrow O(n)$**
- **Cache per layer: K and V, each (12, T, 64)**
- **GPT-2 Small: 2 × 12 layers × 768 = 18,432 values per token, 37 KB in fp16**
- **Full 1,024-token context: about 38 MB per sequence**



Toy example: 5 tokens, one layer. Each row is one generation step; counts are K/V computations per layer.

Mixture of Experts

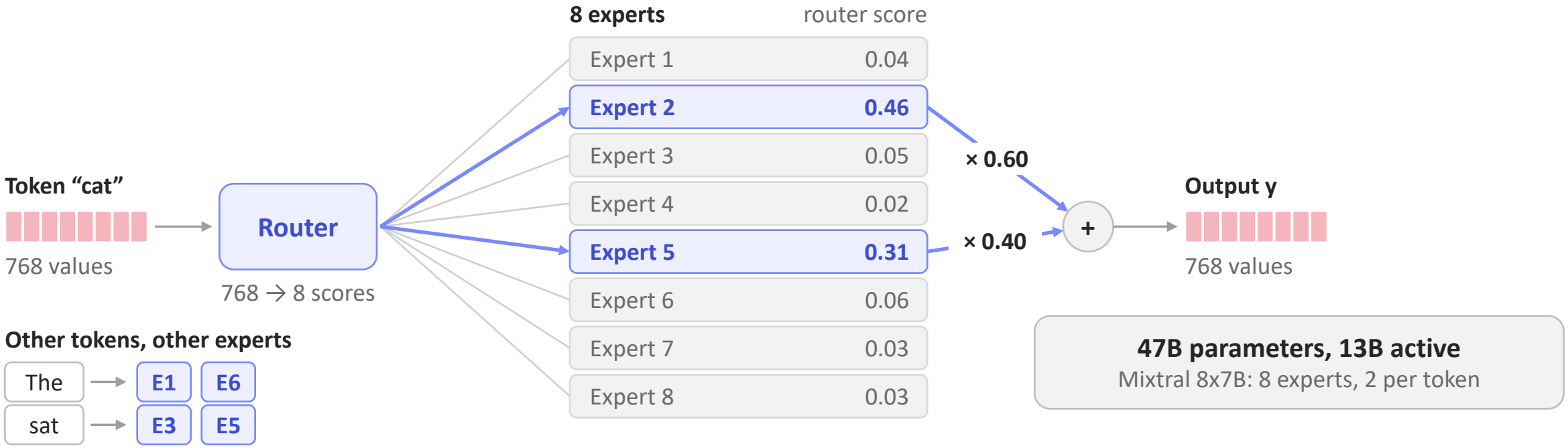
- MoE replaces each block's MLP, two thirds of its parameters, with several expert MLPs
- A router weights all 8 experts; every expert runs for every token
- Experts are learned during training, with the router; expert collapse: a few experts take over



Illustrative routing for one token; scores sum to 1 and weight every expert (line width). Mixtral: Jiang et al., 2024.

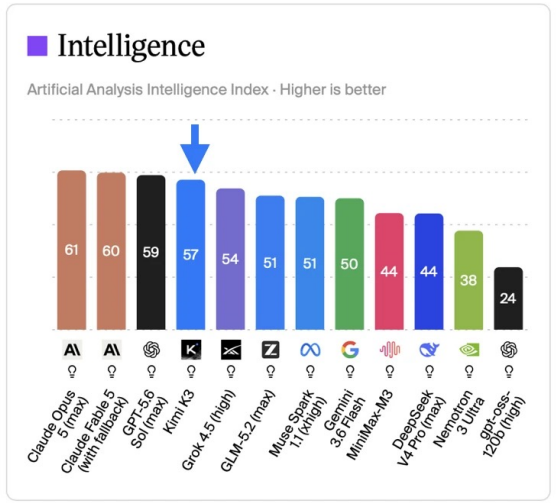
Mixture of Experts

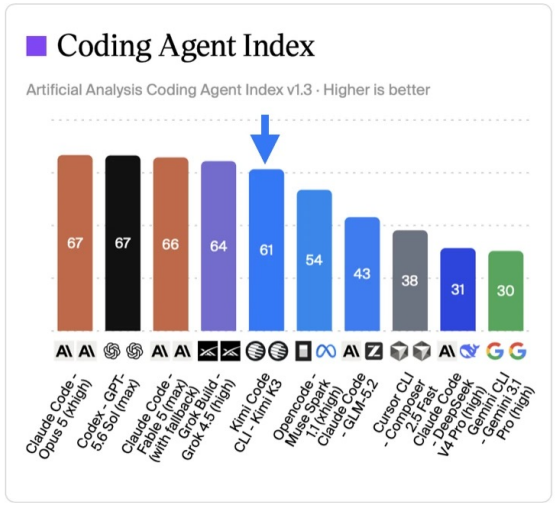
- Sparse MoE activates only a few expert MLPs per token instead of all of them
- A router sends each token to its top 2 experts; the rest stay idle for that token
- Experts are learned during training; expert collapse: the router favors a few, the rest barely train



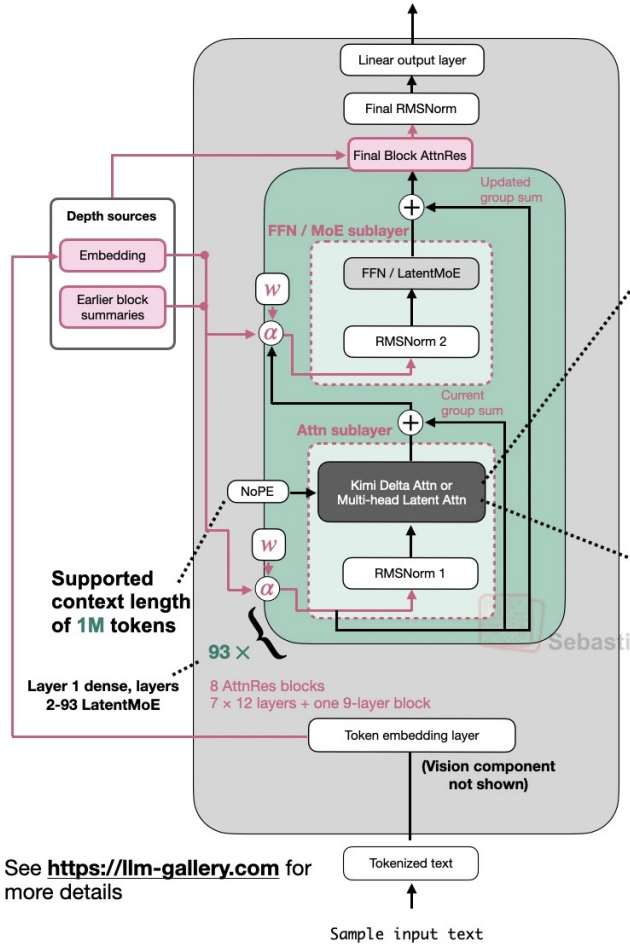
Illustrative routing for one token; gate weights renormalized over the 2 chosen experts. Mixtral: Jiang et al., 2024.

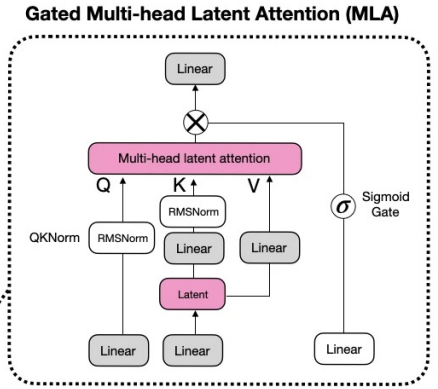
Kimi K3

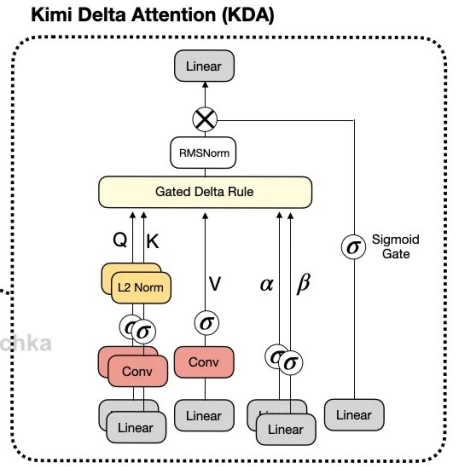




https://sebastianraschka.com/llm-architecture-gallery/#architecture-gallery







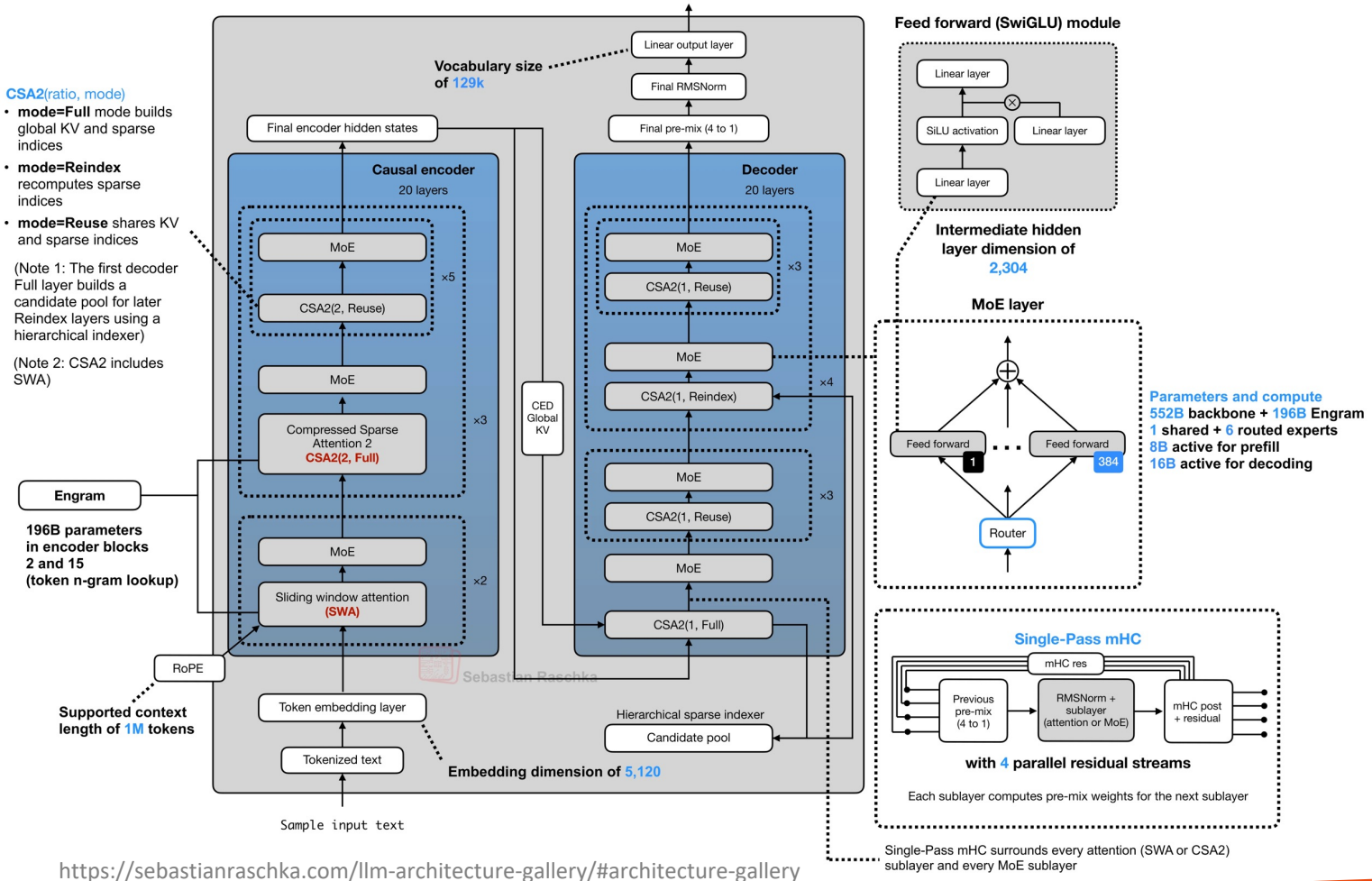
3:1 Ratio

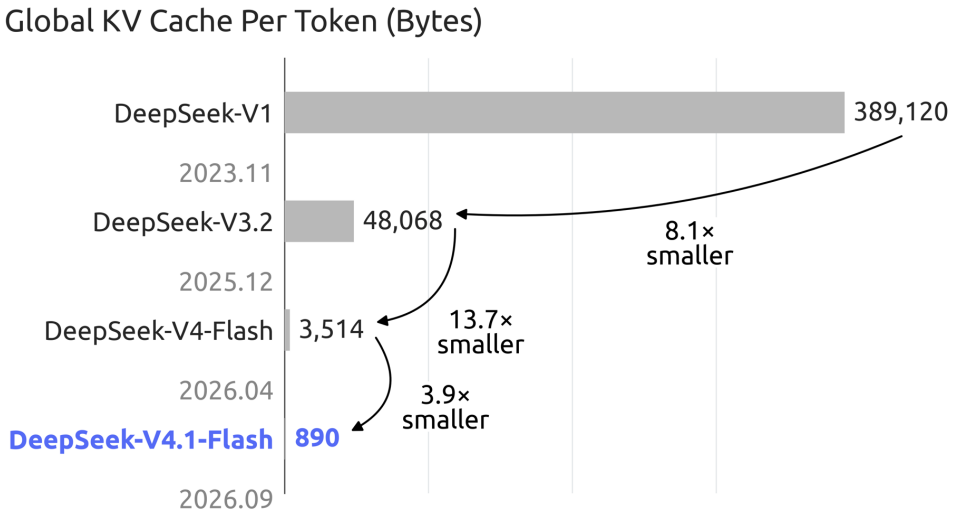
```
Layer 1   : Kimi Delta Attn → MoE
Layer 2   : Kimi Delta Attn → MoE
Layer 3   : Kimi Delta Attn → MoE
Layer 4   : Gated Multi-head Latent Attn → MoE
...
```

DeepSeek-V4.1-Flash

DeepSeek-V4.1-Flash (552B)

Text backbone shown. Vision encoder and DSpark speculative decoding modules omitted.

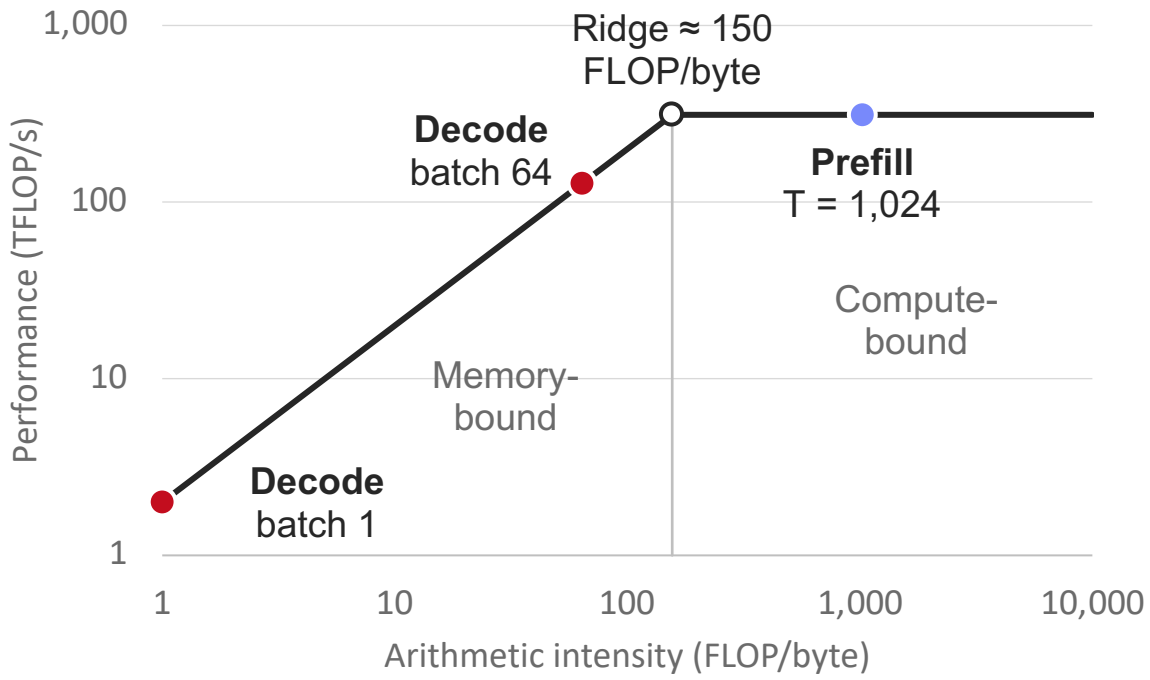




https://sebastianraschka.com/llm-architecture-gallery/#architecture-gallery

Roofline: Compute-Bound vs. Memory-Bound

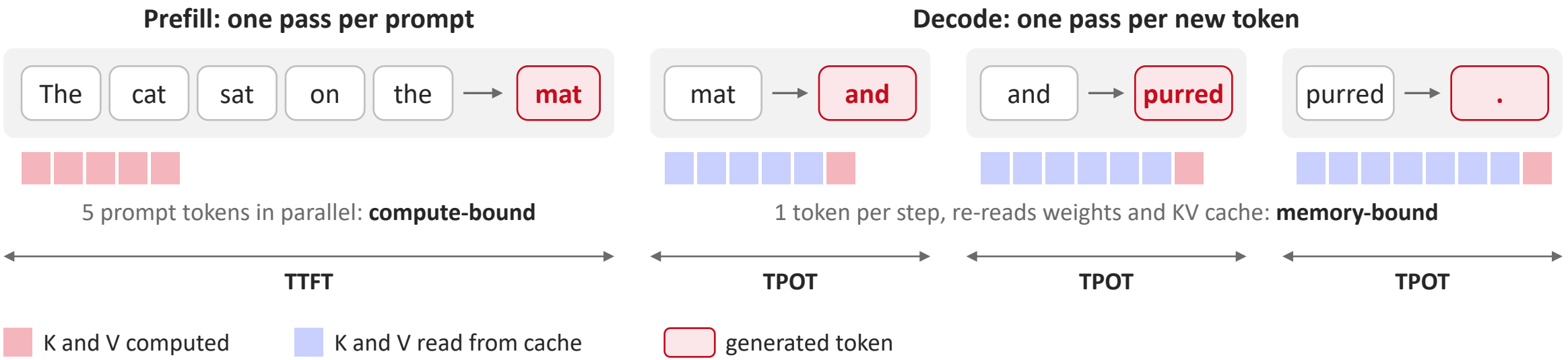
- **Attainable FLOP/s = min(peak compute, memory bandwidth × arithmetic intensity)**
- **Arithmetic intensity: FLOPs per byte read from memory**
- **Ridge point, A100: 312 TFLOP/s ÷ 2 TB/s ≈ 150 FLOP/byte**
- **Decode, batch 1, fp16: about 1 FLOP/byte (2 FLOPs per 2-byte weight), memory-bound**
- **Prefill of T tokens: about T FLOP/byte, compute-bound for long prompts**
- **Batching B requests raises decode intensity to about B FLOP/byte**



Log-log axes. A100 fp16: peak 312 TFLOP/s, bandwidth 2 TB/s. Batch size 64 is illustrative.

Prefill and Decode

- **Prefill: the whole prompt in one forward pass, fills the KV cache**
- **Decode: one new token per step, reads all weights and the KV cache**
- **Prefill sets time to first token (TTFT); decode sets time per output token (TPOT)**
- **Prefill is compute-bound; decode at small batch size is memory-bound**



Project

damsie

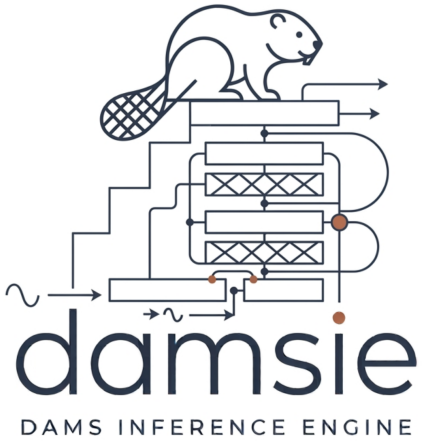
main damsie — Find file — Code

AGENTS.md
Philipp Wilhelm Ortner authored 1 week ago — 34240c43 — History

Name	Last commit	Last update
docs	init	1 week ago
scripts	init	1 week ago
src	init	1 week ago
.clang-format	init	1 week ago
.gitignore	init	1 week ago
AGENTS.md	AGENTS.md	1 week ago
CMakeLists.txt	init	1 week ago
README.md	init	1 week ago
main.cpp	init	1 week ago

README.md — Edit file

Damsie - Educational Inference Engine



Project Machine Learning Systems Assignment 1

End-to-End GPT-2 Inference

The goal of Assignment 1 is to implement a working, end-to-end LLM inference engine for GPT-2 small (`openai-community/gpt2`, 124M parameters) in C++ from scratch. Starting from a minimal C++23 scaffold and a Python conversion script (`scripts/convert.py`) that packs a Hugging Face checkpoint into a single flat binary file, you will implement every stage of the inference pipeline: memory-mapped model loading, Byte-Pair Encoding (BPE) tokenization, the core FP32 tensor kernels, the GPT-2 transformer forward pass, and autoregressive token generation with greedy and stochastic sampling.

Start simple and build incrementally. Rather than tackling every feature and optimization at once, begin with the simplest possible implementation: straightforward nested loops in FP32, no KV-cache (recomputing the full token sequence on each step), and single-token greedy decoding. Once you have a clean, verified end-to-end baseline that produces the exact reference outputs, layer on full autoregressive generation, stochastic sampling, a KV-cache, and further optimizations one step at a time, testing each change against your working baseline.

Getting Started

First, download the pretrained GPT-2 small checkpoint from Hugging Face and convert its weights (`model.safetensors`), configuration (`config.json`), and vocabulary (`vocab.json`, `merges.txt`) into a single memory-mappable binary file:

```
hf download openai-community/gpt2 --local-dir models/gpt2
python3 scripts/convert.py models/gpt2 gpt2.bin
```

Next, configure and build the C++ scaffold with CMake and Ninja, and run the binary by passing the converted model file and a text prompt:

```
cmake -B build -G Ninja -S .
ninja -C build
./build/damsie gpt2.bin "Alan Turing theorized that computers would"
```

Pass `-DCMAKE_BUILD_TYPE=Debug` to `cmake` to build with debug symbols during development.

What You Need to Implement

Project

- **Read the assignment description**
- **Get started with the implementation**
  - If you have questions: mail or forum
  - Feedback: `AGENTS.md`
- **Write your own tests (C++ unit tests & Python reference checks)**
- **Add a KV cache**
  - Where? How? Data structure? Allocation? Memory ownership? Masking?
- **No required reading, will publish a list of resources**
- **No interview**
- **A2 Topics next session, bring your own ideas**

Questions