CS 668 Applied Large Language Models
Spring Semester, 2026

Doc 21 GPRO & LLM Architecture
Arp 9, 2026

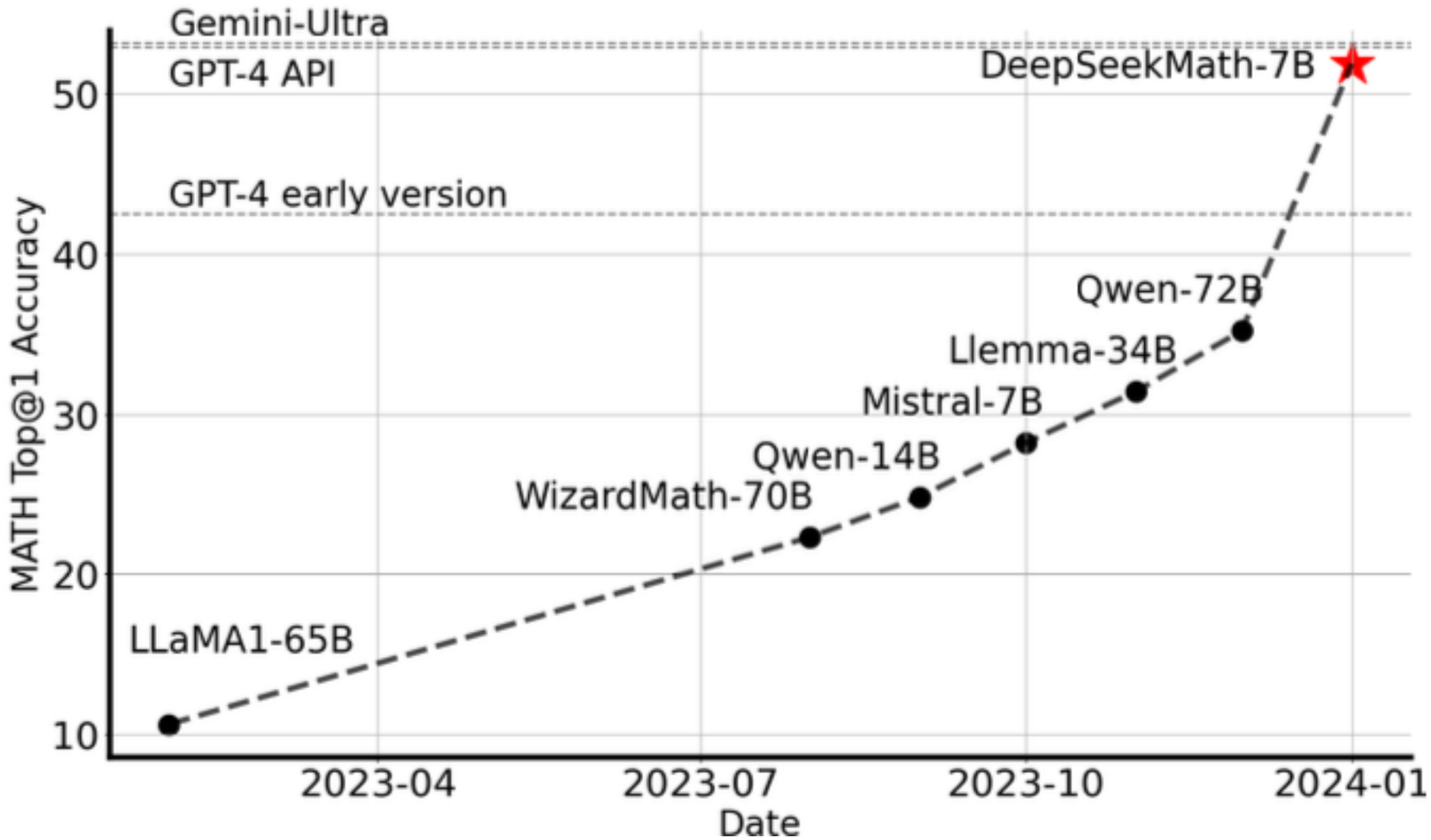
TurboQuant

DeepSeekMath & The Big Improvement

Math is deterministic

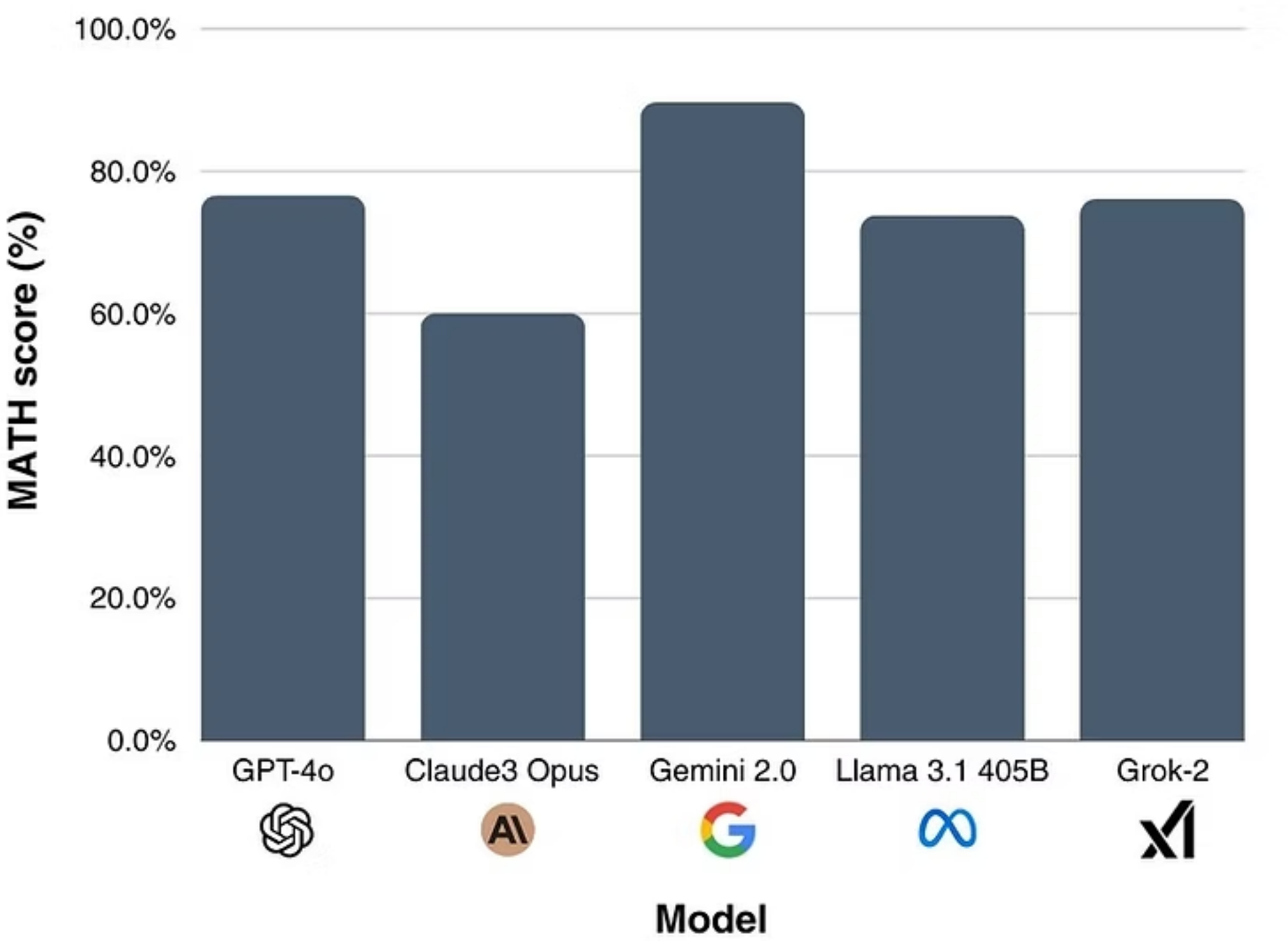
LLMs produce what a good answer probably looks like

DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models
27 Apr 2024


Gemini-Ultra
GPT-4 API
GPT-4 early version
DeepSeekMath-7B
Qwen-72B
Llemma-34B
Mistral-7B
Qwen-14B
WizardMath-70B
LLaMA1-65B
MATH Top@1 Accuracy
50
40
30
20
10
2023-04
2023-07
2023-10
2024-01
Date

Best LLM for the MATH benchmark

Comparing the main frontier models on the MATH benchmark.



Last updated: December, 2024

https://www.bracai.eu/post/llm-math-benchmark

	QwQ 32B-preview	OpenAI o1-preview	OpenAI o1-mini	GPT-4o	Claude3.5 Sonnet	Qwen2.5-72B Instruct
GPQA Pass@1	65.2	72.3	60.0	53.6	65.0	49.0
AIME Pass@1	50.0	44.6	56.7	9.3	16.0	23.3
MATH-500 Pass@1	90.6	85.5	90.0	76.6	78.3	82.6
LiveCodeBench 2024.08-2024.11	50.0	53.6	58.0	33.4	36.3	30.4

https://qwenlm.github.io/blog/qwq-32b-preview/

MATH LLM Benchmark

12,500 problems sourced from high school math competitions

A PhD student without a strong math background scored 40%

A three-time IMO gold medalist scored 90% (IMO = International Mathematical Olympiad)

https://www.bracai.eu/post/llm-math-benchmark

Cagney can frost a cupcake every 20 seconds and Lacey can frost a cupcake every 30 seconds. Working together, how many cupcakes can they frost in 5 minutes?

A)10. B) 15 C) 20 D) 25 E 30

A taxi ride costs $\$1.50$ plus $\$0.25$ per mile traveled. How much does a 5-mile taxi ride cost?

(A) \$2.25 (B) \$2.50 (C) \$2.75 (D) \$3.00 (E) \$3.25

What value of x satisfies

$$x - \frac{3}{4} = \frac{5}{12} - \frac{1}{3}?$$

(A) $-\frac{2}{3}$ (B) $\frac{7}{36}$ (C) $\frac{7}{12}$ (D) $\frac{2}{3}$ (E) $\frac{5}{6}$

What is the value of

$$3 + \frac{1}{3 + \frac{1}{3 + \frac{1}{3}}}?$$

(A) $\frac{31}{10}$ (B) $\frac{49}{15}$ (C) $\frac{33}{10}$ (D) $\frac{109}{33}$ (E) $\frac{15}{4}$

The median of the list

$n, n+3, n+4, n+5, n+6, n+8, n+10, n+12, n+15$

is 10. What is the mean?

(A) 4 (B) 6 (C) 7 (D) 10 (E) 11

A number x is 2 more than the product of its reciprocal and its additive inverse. In which interval does the number lie?

(A) $-4 \leq x \leq -2$ (B) $-2 < x \leq 0$ (C) $0 < x \leq 2$
(D) $2 < x \leq 4$ **(E)** $4 < x \leq 6$

In the year 2001, the United States will host the International Mathematical Olympiad. Let I, M, and O be distinct positive integers such that the product $I \cdot M \cdot O = 2001$. What's the largest possible value of the sum $I + M + O$?

DeepSeekMath - Math Pre-Training at Scale

DeepSeekMath Corpus - 120B tokens

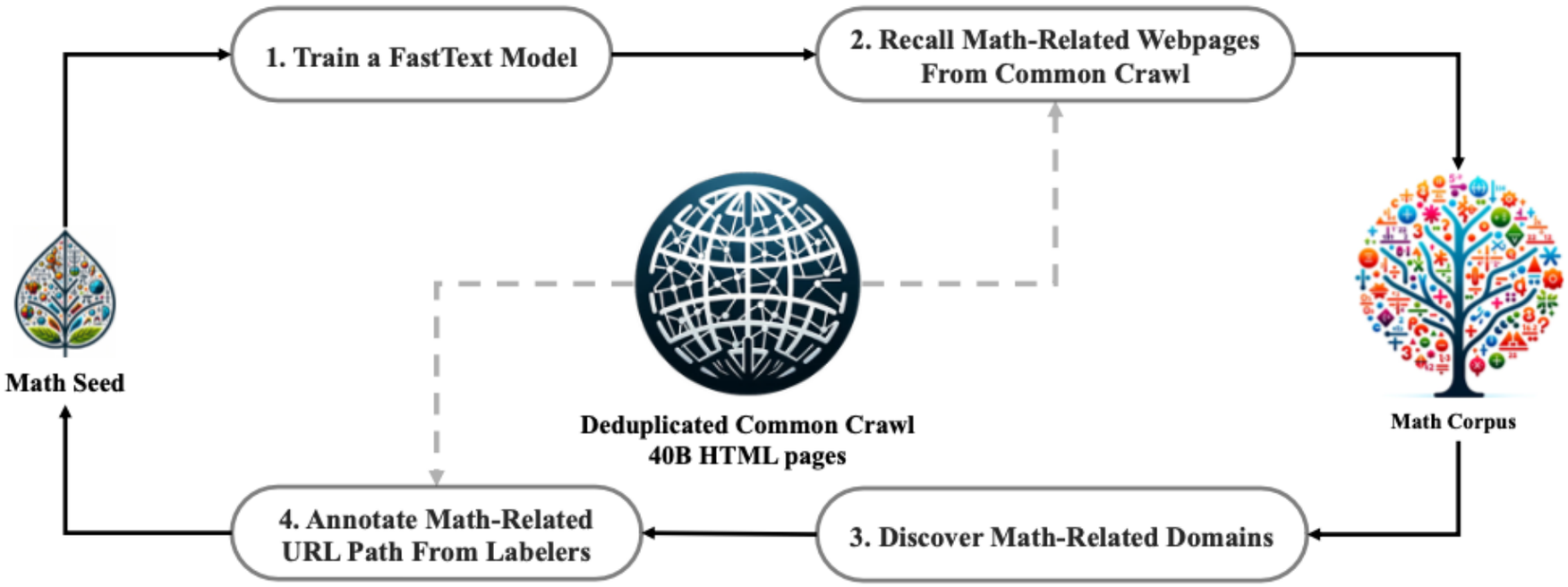


Figure 2 | An iterative pipeline that collects mathematical web pages from Common Crawl.

OpenWebMath initial seed corpus

500,000 data points from the seed corpus as positive training examples
500,000 web pages from Common Crawl as negative ones.

Common Crawl

2 billion web pages in each of its monthly crawls,
generating around 250 terabytes of data each month

Stored on Amazon Web Services' Public Data Sets

Access to the corpus hosted by Amazon is free.

https://commoncrawl.org/overview

DeepSeek reduced the Common Crawl to 40B webpages

URL-based deduplication and near-deduplication techniques,

FastText Model

Text classification model

To train the model, they used
vector dimension to 256,
learning rate to 0.1,
maximum length,
of word n-gram to 3,
the minimum number of word occurrences to 3
the number oftraining epochs to 3

Rank the collected pages according to fastText model
Only preserve the top-ranking ones

Collecting the Corpus

First iteration of data collection missed Math webpages

Enriched the seed corpus

- Calculate the percentage of Math pages in a domain
- If 10% of the pages are Math pages, manually annotate the URLs
- Web pages linked to these URLs, yet uncollected, will be added to the seed corpus

Retrain model and repeat

After 4 iterations 98% of data was collected in third iteration so stopped

35.5M mathematical web pages, totaling 120B tokens

Benchmark Contamination

Need to remove answers to Math benchmarks questions from training data

- GSM8K
- MATH (12,500 problems)
- Chinese benchmarks such as
- CMATH
- AGIEval

Filtering criteria

Any text segment containing a 10-gram string that matches exactly with any sub-string from the evaluation benchmarks is removed from our math training corpus.

For benchmark texts that are shorter than 10 grams but have at least 3 grams

employ exact matching to filter out contaminated web pages

Validating the Quality of the DeepSeekMath Corpus

MathPile(8.9B tokens)

Textbooks, Wikipedia, ProofWiki, CommonCrawl, StackExchange, and arXiv

OpenWebMath

CommonCrawl, 13.6B tokens

Proof-Pile-2

OpenWeb-Math, AlgebraicStack (10.3B tokens), and arXiv papers (28.0B tokens).

Train DeepSeek-LLM 1.3B separately on each of the Corpusora

Validating the Quality of the DeepSeekMath Corpus

Train DeepSeek-LLM 1.3B separately on each of the Corpusora

AdamW optimizer with
$\beta 1 = 0.9$, $\beta 2 = 0.95$, and weight_decay= 0.1

Multi-step learning rate schedule
Learning rate reaches the peak after 2,000 warmup steps,
Decreases to its 31.6% after 80% of the training process,
Further decreases to 10.0% of the peak after 90% of the training process.

Maximum learning rate 5.3e-4,

Batch size of 4M tokens

4K context length

Validating the Quality of the DeepSeekMath Corpus

Math Corpus	Size	English Benchmarks					Chinese Benchmarks		
		GSM8K	MATH	OCW	SAT	MMLU STEM	CMATH	Gaokao MathCloze	Gaokao MathQA
No Math Training	N/A	2.9%	3.0%	2.9%	15.6%	19.5%	12.3%	0.8%	17.9%
MathPile	8.9B	2.7%	3.3%	2.2%	12.5%	15.7%	1.2%	0.0%	2.8%
OpenWebMath	13.6B	11.5%	8.9%	3.7%	31.3%	29.6%	16.8%	0.0%	14.2%
Proof-Pile-2	51.9B	14.3%	11.2%	3.7%	43.8%	29.2%	19.9%	5.1%	11.7%
DeepSeekMath Corpus	**120.2B**	**23.8%**	**13.6%**	**4.8%**	**56.3%**	**33.1%**	**41.5%**	**5.9%**	**23.6%**

Table 1 | Performance of DeepSeek-LLM 1.3B trained on different mathematical corpora, evaluated using few-shot chain-of-thought prompting. Corpus sizes are calculated using our tokenizer with a vocabulary size of 100K.

DeepSeekMath-Base 7B

Start with DeepSeek-Coder-Base-v1.5 7B

Trained for 500B tokens

- 56% from DeepSeekMath Corpus,
- 4% from AlgebraicStack,
- 10% from arXiv,
- 20% is Github code,
- 10% is natural language data from Common Crawl in both English and Chinese.

AdamW optimizer with

- $\beta 1 = 0.9$, $\beta 2 = 0.95$, and weight_decay= 0.1

Multi-step learning rate schedule

- Learning rate reaches the peak after 2,000 warmup steps,
- Decreases to its 31.6% after 80% of the training process,
- Further decreases to 10.0% of the peak after 90% of the training process.

Maximum learning rate 4.3e-4,

Batch size of 10M tokens

4K context length

DeepSeekMath-Instruct 7B

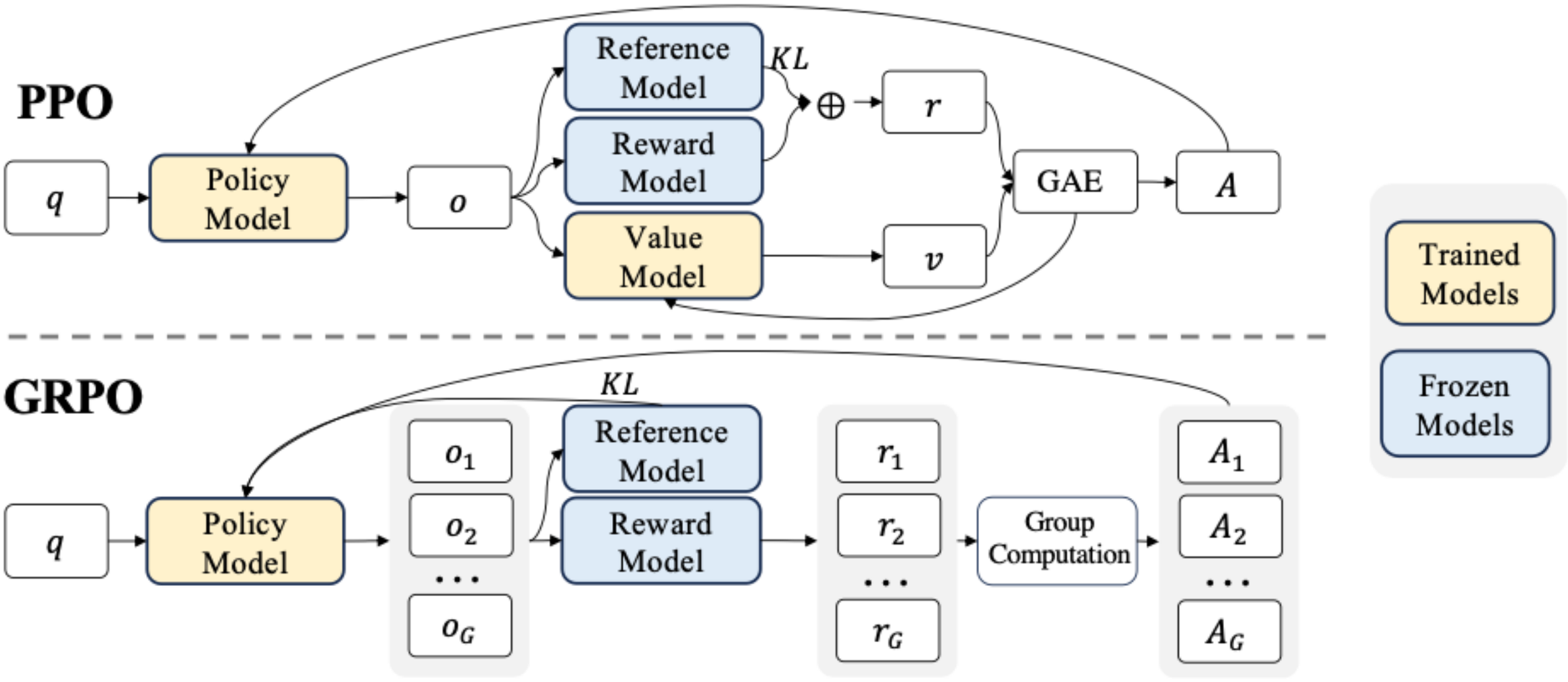
Start with DeepSeekMath-Base

Training examples are randomly concatenated to 4K tokens.
500 steps
Batch size of 256
Constant learning rate of 5e-5

DeepSeek-Coder-Base-v1.5 7B -> DeepSeekMath-Base -> DeepSeekMath-Instruct 7B

Now Use Reinforcement Learning

Group Relative Policy Optimization (GRPO)



GRPO Basics

Sampling:

Generate multiple outputs for each prompt using the current policy

Reward Scoring:

Each generation is scored using a reward function, could be rule-based or outcome-based

Advantage Calculation:

The average reward of the generated outputs is used as a baseline. The advantage of each solution within the group is then computed relative to this baseline. The reward is normalized within a group.

Policy Optimization:

The policy tries to maximize the GRPO objective, which includes the calculated advantages and a KL divergence term.

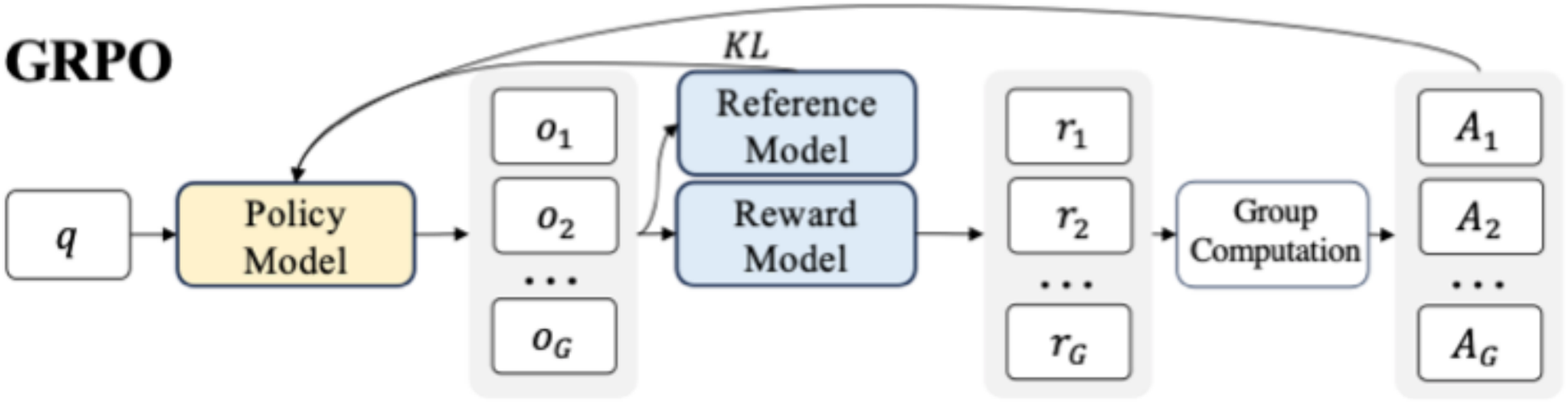
KL (Kullback-Leibler) divergence

Measure of how one probability distribution differs from another

In PPO quantifies the difference between
Current policy (the policy being updated) and
Old policy

Policy Stability:
PPO update policies gradually
The KL divergence helps gauge this difference

Trust Region Control:
By measuring how far the new policy has moved from the old one, PPO can enforce a “trust region” that prevents overly large updates, ensuring stable training.



Algorithm 1 Iterative Group Relative Policy Optimization

Input initial policy model $\pi_{\theta_{\text{init}}}$; reward models r_φ; task prompts $\mathcal{D}$; hyperparameters ε, β, μ

policy model $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$
for iteration = 1, . . . , I **do**
    reference model $\pi_{ref} \leftarrow \pi_\theta$
    for step = 1, . . . , M **do**
        Sample a batch $\mathcal{D}_b$ from $\mathcal{D}$
        Update the old policy model $\pi_{\theta_{old}} \leftarrow \pi_\theta$
        Sample G outputs $\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(\cdot \mid q)$ for each question $q \in \mathcal{D}_b$
        Compute rewards $\{r_i\}_{i=1}^{G}$ for each sampled output o_i by running r_φ
        Compute $\hat{A}_{i,t}$ for the t-th token of o_i through group relative advantage estimation.
        for GRPO iteration = 1, . . . , μ **do**
            Update the policy model π_θ by maximizing the GRPO objective (Equation 21)
    Update r_φ through continuous training using a replay mechanism.

Output π_θ

DeepSeekMath-RL

DeepSeek-Coder-Base-v1.5 7B -> DeepSeekMath-Base
-> DeepSeekMath-Instruct 7B
-> DeepSeekMath-RL

GRPO and the Countdown Game

https://www.philschmid.de/mini-deepseek-r1

https://github.com/philschmid/deep-learning-pytorch-huggingface/blob/main/training/mini-deepseek-r1-aha-grpo.ipynb

Countdown game

Use set numbers and basic arithmetic operations (+, -, ×, ÷) to reach or get as close as possible to a target number.

Target Number: 952
Available Numbers: 25, 50, 75, 100, 3, 6

(100 × (3 × 3)) + (50 + 6 / 3) = 952

Training Dataset

```
from transformers import AutoTokenizer
from datasets import load_dataset

dataset_id = "Jiayi-Pan/Countdown-Tasks-3to4"
dataset = load_dataset(dataset_id, split="train")
dataset = dataset.shuffle(seed=42).select(range(50000))
```

```
{'target': 88, 'nums': [95, 21, 3]}
{'target': 62, 'nums': [22, 29, 69]}
{'target': 17, 'nums': [9, 73, 38, 2]}
{'target': 98, 'nums': [33, 4, 2, 59]}
{'target': 53, 'nums': [25, 42, 13, 83]}
{'target': 19, 'nums': [15, 23, 57]}
{'target': 47, 'nums': [2, 37, 5]}
{'target': 71, 'nums': [74, 39, 66, 81]}
{'target': 26, 'nums': [98, 58, 47, 33]}
{'target': 83, 'nums': [67, 65, 87, 38]}
```

```
tokenizer = AutoTokenizer.from_pretrained(“Qwen/Qwen2.5-3B-Instruct")

def generate_r1_prompt(numbers, target):
    r1_prefix = [{
        "role": "system",
        "content": "You are a helpful assistant. You first thinks about the reasoning process in the mind and then provides the user with the answer."
      },
      {
        "role": "user",
        "content": f"Using the numbers {numbers}, create an equation that equals {target}. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Show your work in <think> </think> tags. And return the final equation and answer in <answer> </answer> tags, for example <answer> (1 + 2) / 3 = 1 </answer>."
      },
      {
        "role": "assistant",
        "content": "Let me solve this step by step.\n<think>"
      }]
    return {"prompt": tokenizer.apply_chat_template(r1_prefix, tokenize=False,
                                                    continue_final_message=True), "target": target}
```

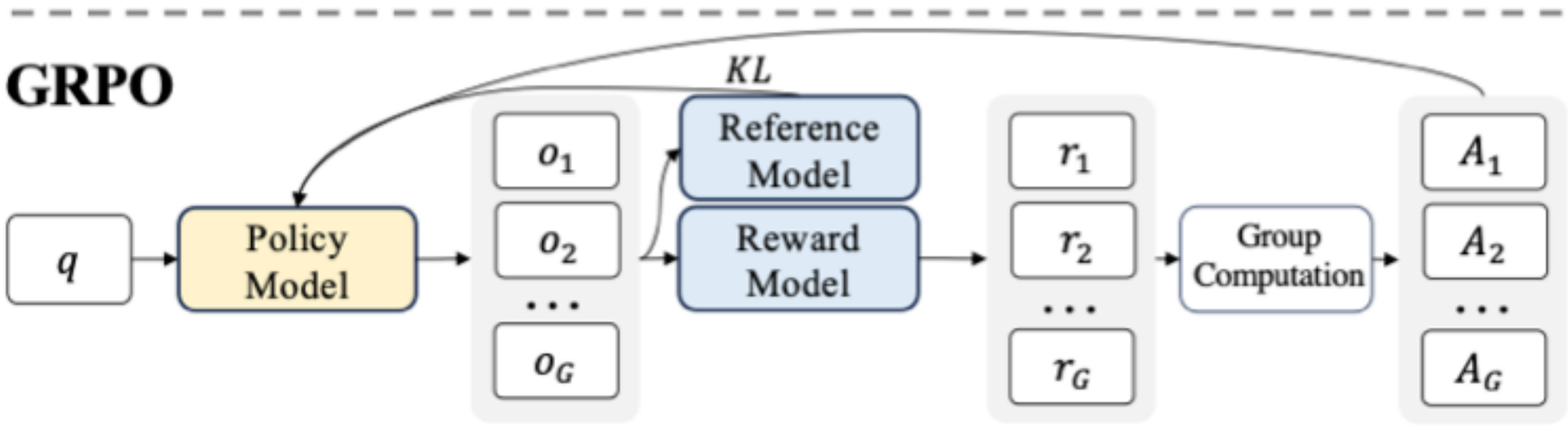
```
# convert our dataset to the r1 prompt
dataset = dataset.map(lambda x: generate_r1_prompt(x["nums"], x["target"]))

# split the dataset into train and test
train_test_split = dataset.train_test_split(test_size=0.1)

train_dataset = train_test_split["train"]
test_dataset = train_test_split["test"]
```

Reward Functions

GRPOTrainer needs reward function (s) to evaluate model

Use

- Properly formatted response
- Correct equation
  - Numbers used only once
  - Math is done correctly



Train the model using GRPO

```
import re

def format_reward_func(completions, target, **kwargs):
    """Format: <think>...</think><answer>...</answer>"""
    rewards = []

    for completion, gt in zip(completions, target):
      try:
        completion = "<think>" + completion
        # Check if the format is correct
        regex = r"^<think>([^<]*(?:<(?!/?think>)[^<]*)*)<\/think>\n<answer>([\s\S]*?)<\/answer>$"

        match = re.search(regex, completion, re.DOTALL)
        # if the format is not correct, reward is 0
        if match is None or len(match.groups()) != 2:
            rewards.append(0.0)
        else:
            rewards.append(1.0)
      except Exception:
        rewards.append(0.0)
    return rewards
```

```
def equation_reward_func(completions, target, nums, **kwargs):
    rewards = []
    for completion, gt, numbers in zip(completions, target, nums):
      try:
       completion = "<think>" + completion
        match = re.search(r"<answer>(.*?)<\/answer>", completion)    # Is format correct
        if match is None:
            rewards.append(0.0)
            continue
        equation = match.group(1).strip()
        used_numbers = [int(n) for n in re.findall(r'\d+', equation)]

        if sorted(used_numbers) != sorted(numbers): # Check if all numbers are used exactly once
            rewards.append(0.0)
            continue
      allowed_pattern = r'^[\d+\-*/().\s]+$'
      if not re.match(allowed_pattern, equation):
        rewards.append(0.0)
        continue

      result = eval(equation, {"__builtins__": None}, {}) # Evaluate the equation
      if abs(float(result) - float(gt)) < 1e-5: # Is equation correct
          rewards.append(1.0)
      else:
          rewards.append(0.0)
    except Exception:
          rewards.append(0.0)
  return rewards
```

```
correct_sample_1 = """We need to find an equation using the numbers 19, 36, 55, and 7
exactly once, with basic arithmetic operations, that equals 65. One possible
combination is 55 + 36 - 19 + 7... </think>
<answer> 55 + 36 - 7 - 19 </answer>"""

correct_sample_2 = """ ... </think>
<answer> 55 + 36 - 7 - 19 </answer>"""

wrong_format = """User: Using the numbers [19, 36, 55, 7], create an equation that equals 65."""

wrong_format_2 = """To find the equation that equals 79 using the numbers 95, 78, 6, 88, I'll start by add
and 95:
95 + 88 = 183
Now, let's subtract 104 from 183 to get 79:
183 - 104 = 79
<think> 183 - 104 = 79 </think><think> 183 - 104 = 79 </think><answer> 183 - 104 = 79 </answer>"""

wrong_result = """ ... </think>
<answer> 55 + 36 - 7 - 18 </answer>"""



test_rewards = format_reward_func(completions=[correct_sample_1, correct_sample_2, wrong_format,
wrong_format_2, wrong_result], target=["65", "65", "65", "65", "65"], nums=[[19, 36, 55, 7]] * 5)
assert test_rewards == [1.0, 1.0, 0.0, 0.0, 1.0], "Reward function is not working"
```

```
test_rewards = format_reward_func(
    completions=[correct_sample_1, correct_sample_2, wrong_format, wrong_format_2, wrong_result],
    target=["65", "65", "65", "65", "65"],
    nums=[[19, 36, 55, 7]] * 5)

assert test_rewards == [1.0, 1.0, 0.0, 0.0, 1.0], "Reward function is not working”
```

```
test_rewards = equation_reward_func(
    completions=[correct_sample_1, correct_sample_2, wrong_format, wrong_format_2, wrong_result],
    target=["65", "65", "65", "65", "65"],
    nums=[[19, 36, 55, 7]] * 5)

assert test_rewards == [1.0, 1.0, 0.0, 0.0, 0.0], "Reward function is not working"
```

```
from trl import GRPOConfig, GRPOTrainer, get_peft_config, ModelConfig

# our model we are going to use as policy
model_config = ModelConfig(
    model_name_or_path="Qwen/Qwen2.5-3B-Instruct",
    torch_dtype="bfloat16",
    attn_implementation="flash_attention_2",
    use_peft=True,
    load_in_4bit=True,
)
```

```
# Hyperparameters
training_args = GRPOConfig(
    output_dir="qwen-r1-aha-moment",
    learning_rate=5e-7,
    lr_scheduler_type="cosine",
    logging_steps=10,
    max_steps=100,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=1,
    gradient_checkpointing=True,
    gradient_checkpointing_kwargs={"use_reentrant": False},
    bf16=True,
    # GRPO specific parameters
    max_prompt_length=256,
    max_completion_length=1024, # max length of the generated output for our solution
    num_generations=2,
    beta=0.001,
)
```

```
trainer = GRPOTrainer(
    model=model_config.model_name_or_path,
    reward_funcs=[format_reward_func, equation_reward_func],
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    peft_config=get_peft_config(model_config),
)
```

```
# Train and push the model to the Hub
trainer.train()
# Save model
trainer.save_model(training_args.output_dir)
```

Run Time

On single CPU, GPU 20 minutes per step, but need 450 steps

Using

- vLLM
- DeepSpeed
- 4 H100 80GBs

Took 6 hours

Hyperparameters

Started with DeepSeekMath paper values

- learning rate of 1e-6
- beta (KL coefficient) of 0.04

Unstable training runs after around 150 steps

So used learning rate to 5e-7 and the beta to 0.001

Full Parameter List

```
torch_dtype: bfloat16
attn_implementation: flash_attention_2
bf16: true
tf32: true
output_dir: runs/qwen-2.5-3b-r1-countdown

# Dataset arguments
dataset_id_or_path: Jiayi-Pan/Countdown-Tasks-3to4

# Lora Arguments
# No LoRA is used here

# Training arguments
max_steps: 450
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
gradient_checkpointing: true
gradient_checkpointing_kwargs:
  use_reentrant: false
learning_rate: 5.0e-7 # 1.0e-6 as in the deepseek math paper 5-e7 from https://hijkzzz.notion.site/unraveling-rlhf-and-its-variants-engineering-insights#147d9a33ecc9806090f3d5c749d31f0
lr_scheduler_type: cosine
warmup_ratio: 0.03
# GRPO specific parameters
beta: 0.001 # 0.04 as in the deepseek math paper 0.001 from https://hijkzzz.notion.site/unraveling-rlhf-and-its-variants-engineering-insights#147d9a33ecc9806090f3d5c749d31f05
max_prompt_length: 256
max_completion_length: 1024
num_generations: 8
use_vllm: true
# vllm_device: "cuda:3"
vllm_gpu_memory_utilization: 0.5

# Logging arguments
logging_strategy: steps
logging_steps: 2
report_to:
- tensorboard
save_strategy: "steps"
save_steps: 25
seed: 42
```

Training Observations

~50 steps

the model has learned the correct format <think>...</think>\n<answer>...</answer>.

100 steps

Success rate for solving the equation is around 25%. The model starts to "reason" with words

200 steps the

Performance seems to converge much slower

~40% success rate.

Learns a new "format" where it solves the equation by trying different combinations and reviewing the results,

450 steps

50% success

The performance still improves slowly

Model kept its new format form from step 200.

Successfull Reasoning Sample up to step 200

<think> First, I need to think about the numbers we have: 14, 70, and 80. To get 85, I think 80 might be a good starting point because it's right in between 85. If I subtract or add 80, it won't lead to 85 easily. Next, 70 seems to be a good intermediate number, especially to use in conjunction with 14.

Now, I can think about different combinations using addition, subtraction and multiplication/division. If I take 80 - 70 = 10, which is not needed. So, I need to find a way to use 14. If I subtract 14 from 80, I get 66, which still isn't 85. If I subtract 70 from 80, I get 10 again, this isn't constructive yet.

Multiplication could create numbers larger than the target 85. Let's consider division: If I divide 70 by 14, I get 5. Then if I add this result to 80, I get 85. This combination seems promising: 80 - (70 / 14) = 80 - 5 = 75, this still isn't quite right. Let me correct my approach: Let's use division and then addition. If I take 80 / (70 / 14) → 80 / 5 = 16, and then 80 - 16 = 64, still incorrect. Let me simplify: 80 / (70 / 14) → 80 / 5 = 16, and then 80 - 16 gives 64 again, let me correct: 80 - (70 / 14) → 80 - 5 = 75, but I need 85. Thus I will use the combinations in a slightly different order to insert numbers: I get 80 + (70 / 14) = 80 + 5 = 85</think>

<answer> 80 + (70 / 14) </answer>

Successfull Reasoning Sample between step 200 and 450

<think> We need to use the numbers 39, 16, and 55 with basic arithmetic operations to get 78. Let's try different combinations:

- 55 + 39 - 16 = 84 - 16 = 68 (too low)
- 55 + 16 - 39 = 71 - 39 = 32 (too low)
- 55 - 16 + 39 = 39 + 39 = 78 (this works)

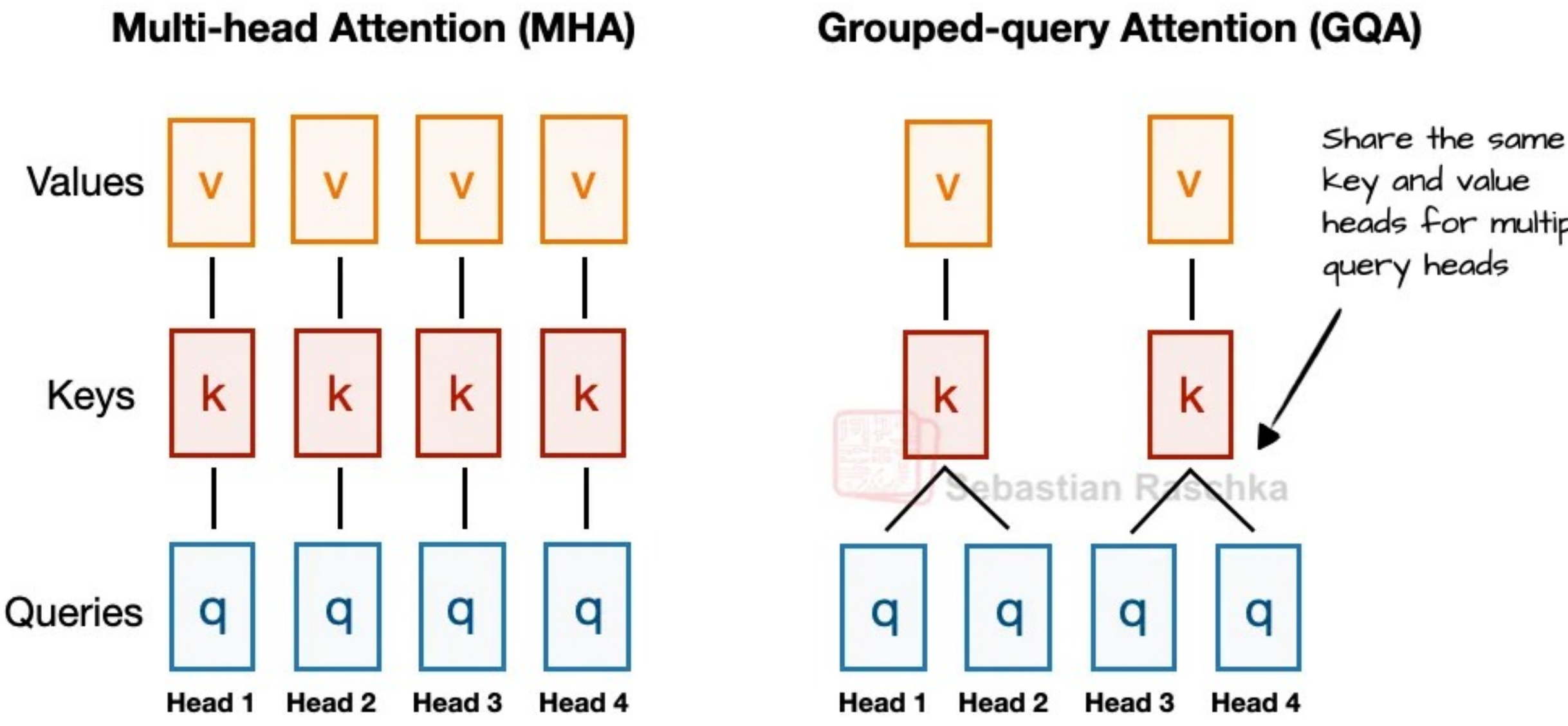
So, the equation is 55 - 16 + 39 = 78. </think>

<answer> 55 - 16 + 39 </answer>

The Big LLM Architecture Comparison

https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison

Grouped-Query Attention (GQA)

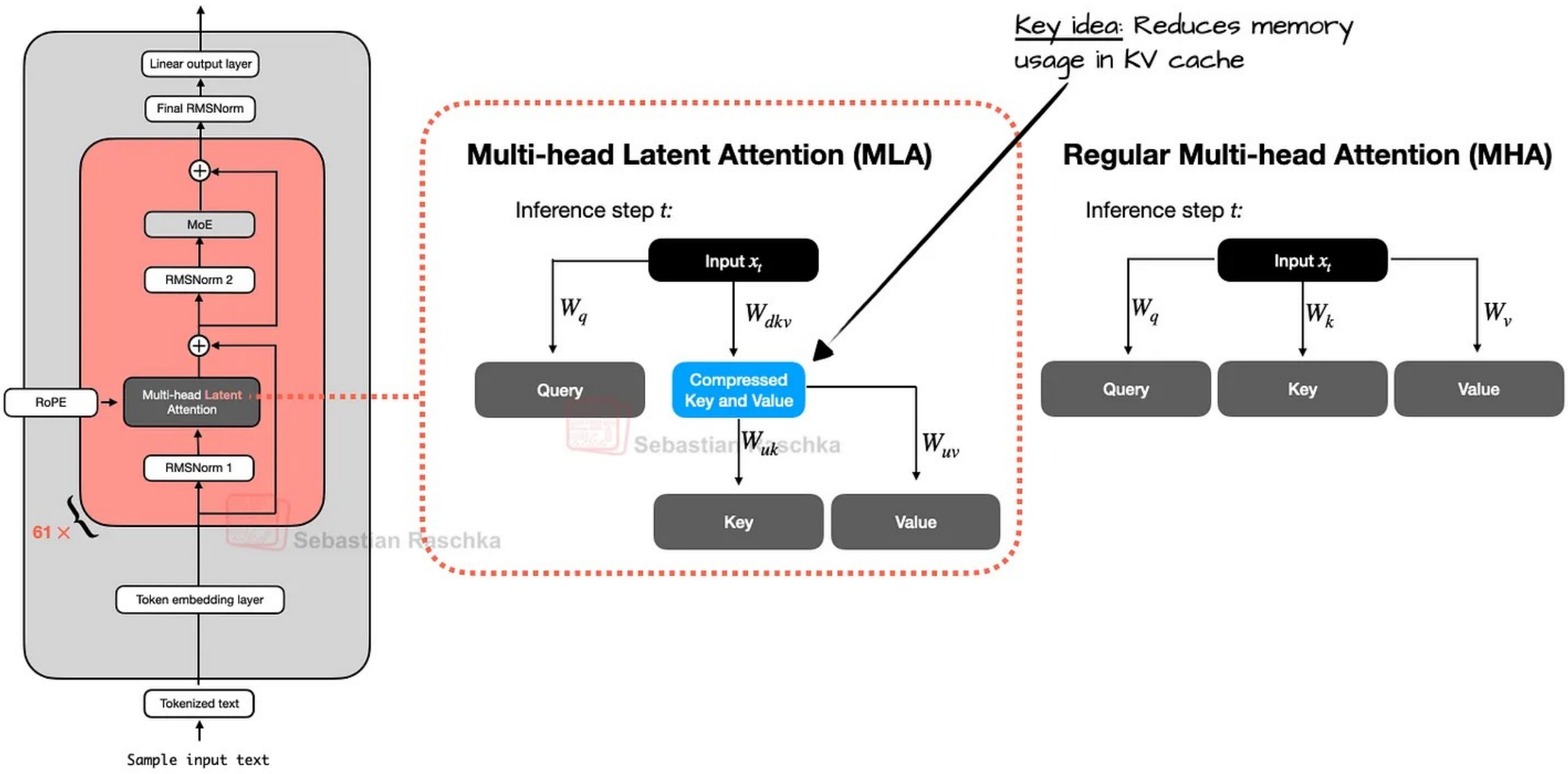


https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison#§8-kimi-k2-and-kimi-k2-thinking

Multi-Head Latent Attention (MLA)

Store compressed Key & Value tensors

Uncompress them when needed



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)

Mixture of Experts (MoE) - Sparce Models

As the parameter count grows

- More information is stored
- Compute time grows

Divide the FFNs into groups - Experts

- 8, 16, 64 experts

A router determines which experts to send a token

Each expert specializes in a domain

How it Works

When a tokne enters an MoE layer

Routing:

The router calculates a probability score for each expert

Selection (Top-K):

The system selects the best experts for that token

Most commonly the Top-2.

Processing:

The token is sent only to those two experts

Weighted Sum:

The outputs from the selected experts are combined and passed to the next layer of the Transformer.

DeepSeek V3

256 experts per MoE module

671 billion parameters

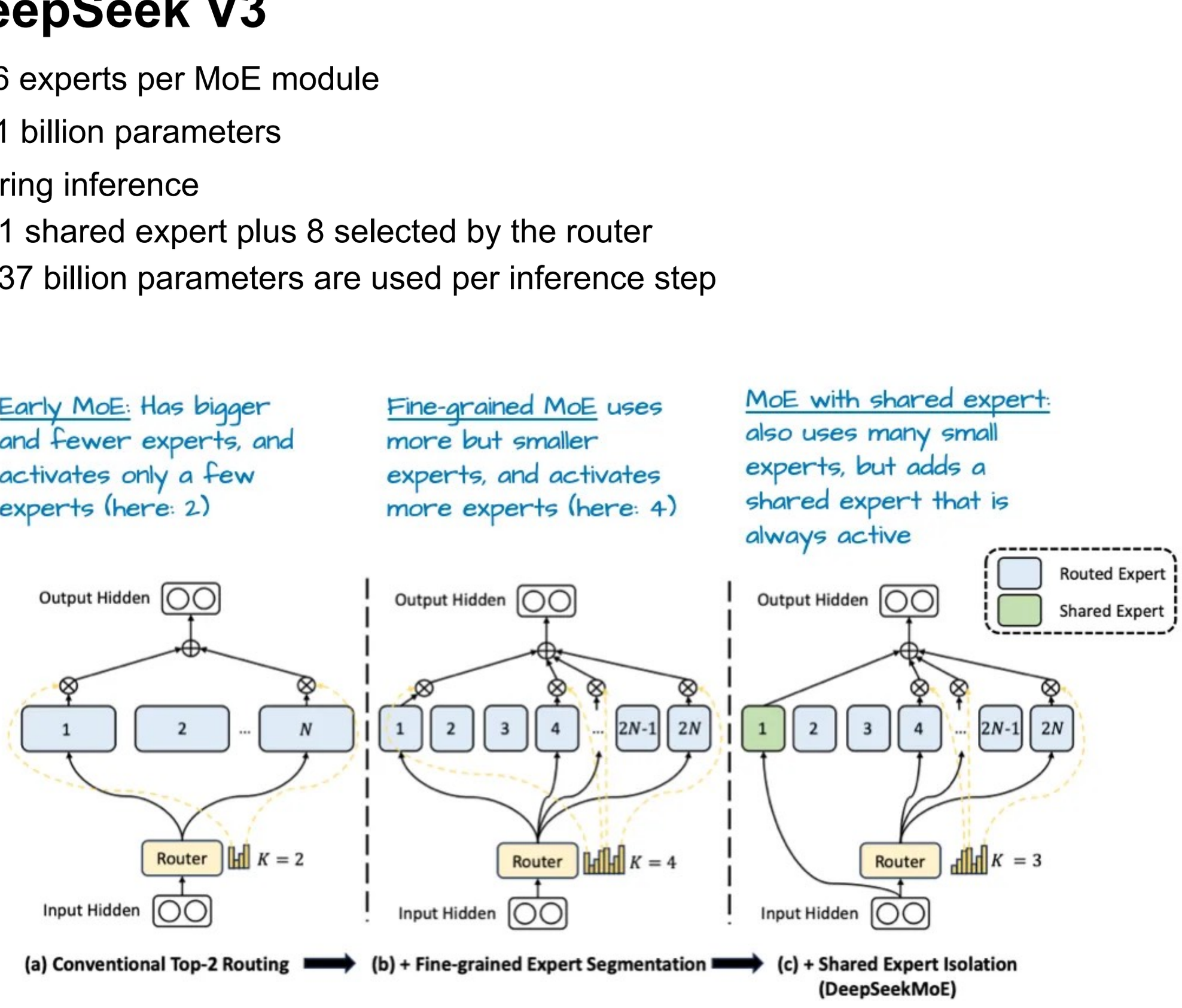
During inference

- 1 shared expert plus 8 selected by the router
- 37 billion parameters are used per inference step

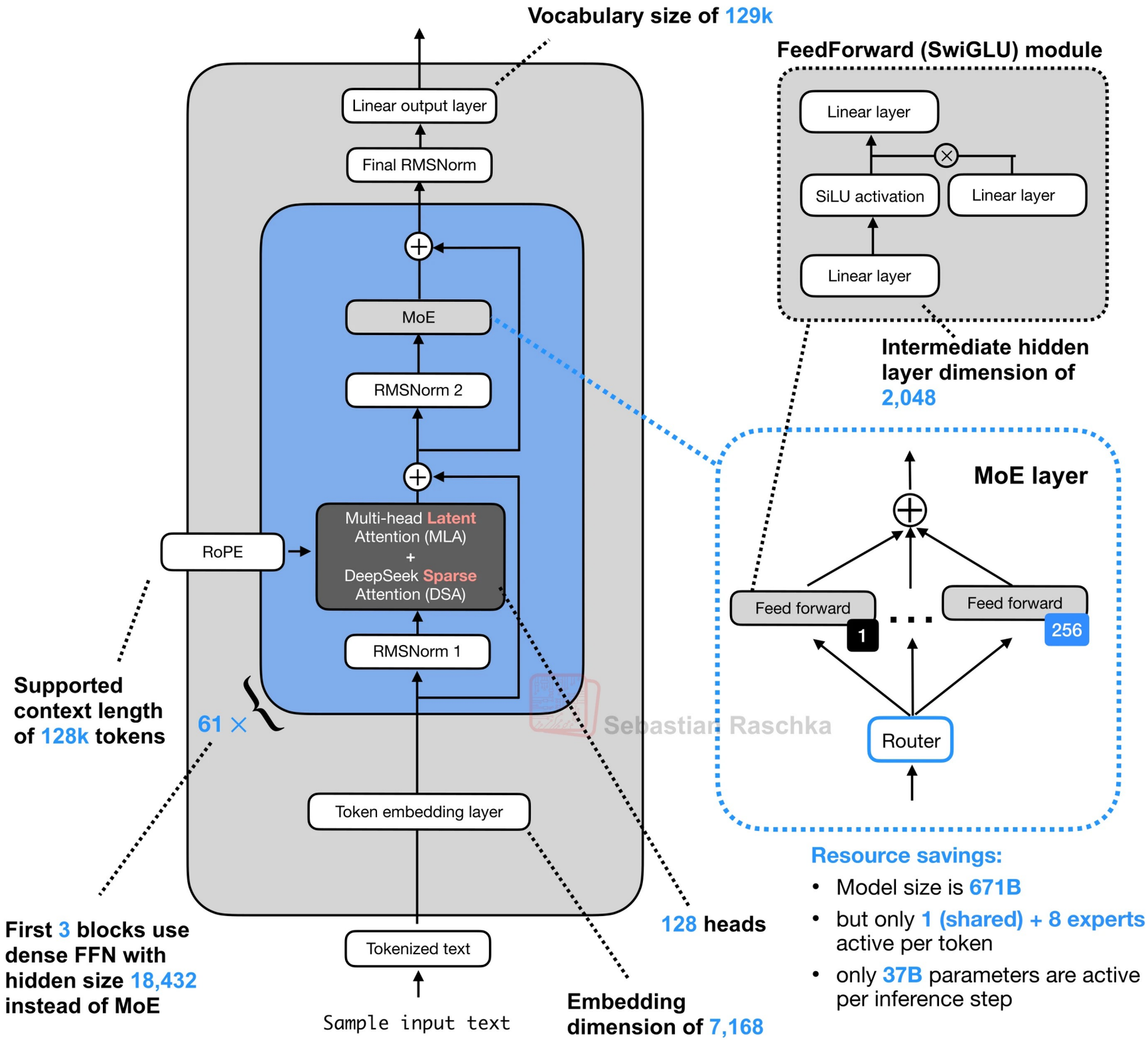
Early MoE: Has bigger and fewer experts, and activates only a few experts (here: 2)

Fine-grained MoE uses more but smaller experts, and activates more experts (here: 4)

MoE with shared expert: also uses many small experts, but adds a shared expert that is always active



DeepSeek V3.2 (671B)

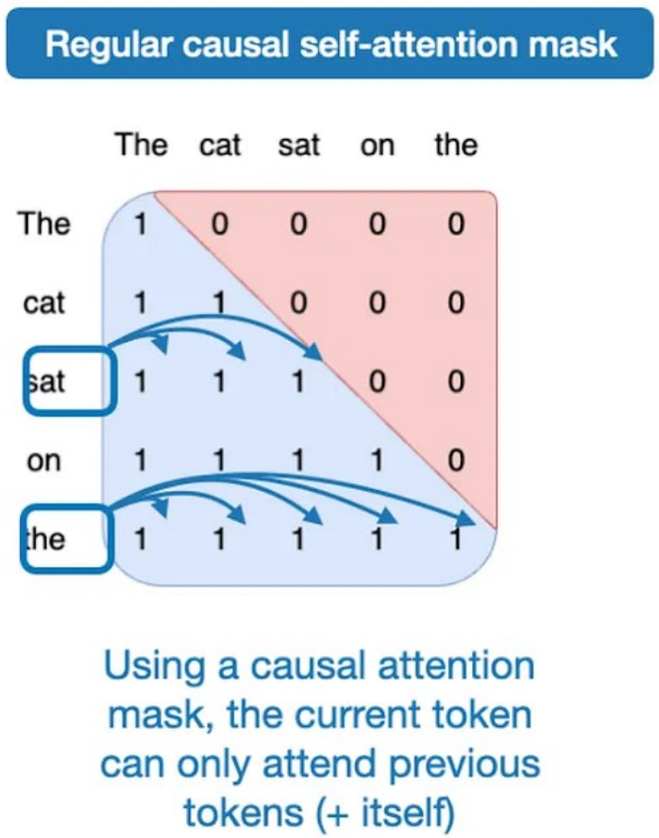


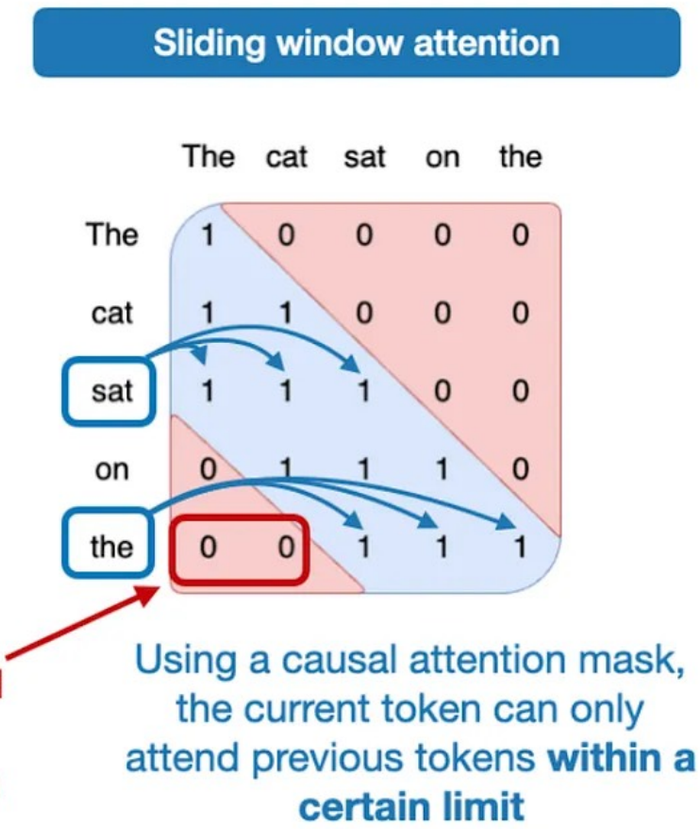
Resource savings:

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

Gemma 3

Uses a Sliding Attention Window to reduce memory - Local Attention





Gemma Sliding Window

Gemma 2

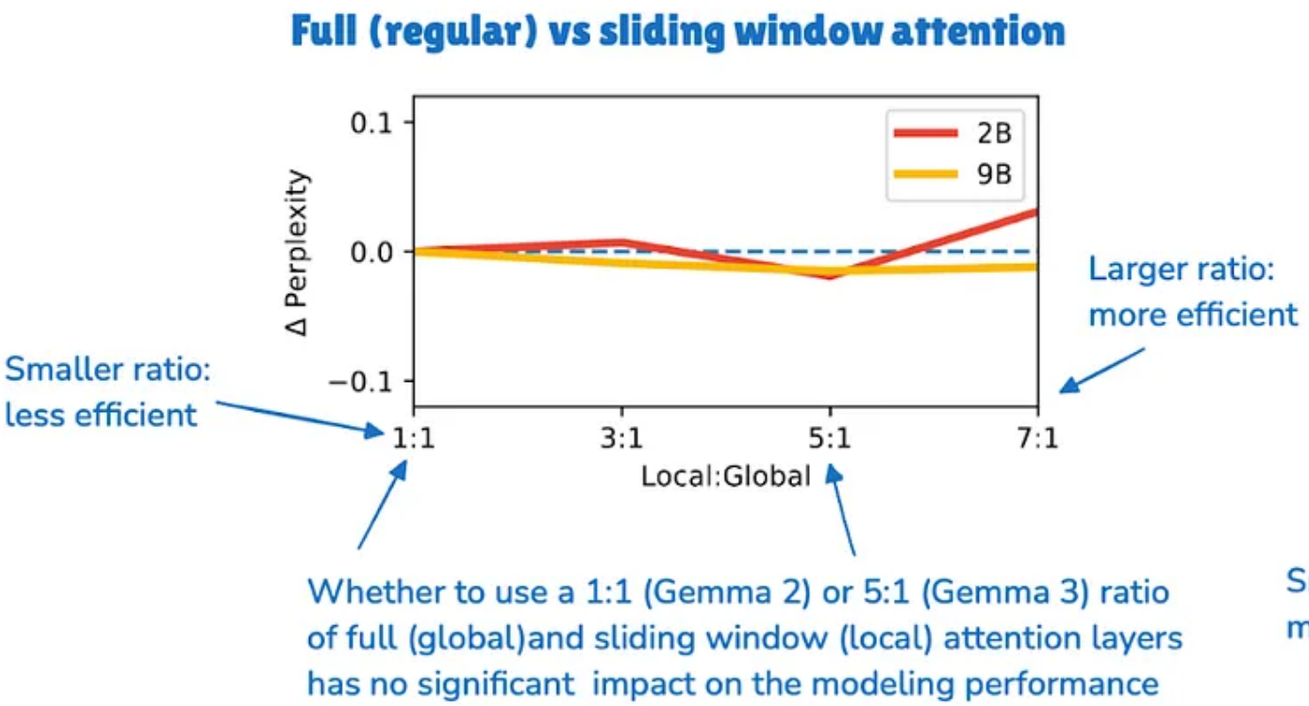
1 full attention layer for each layer using a sliding window 1:1

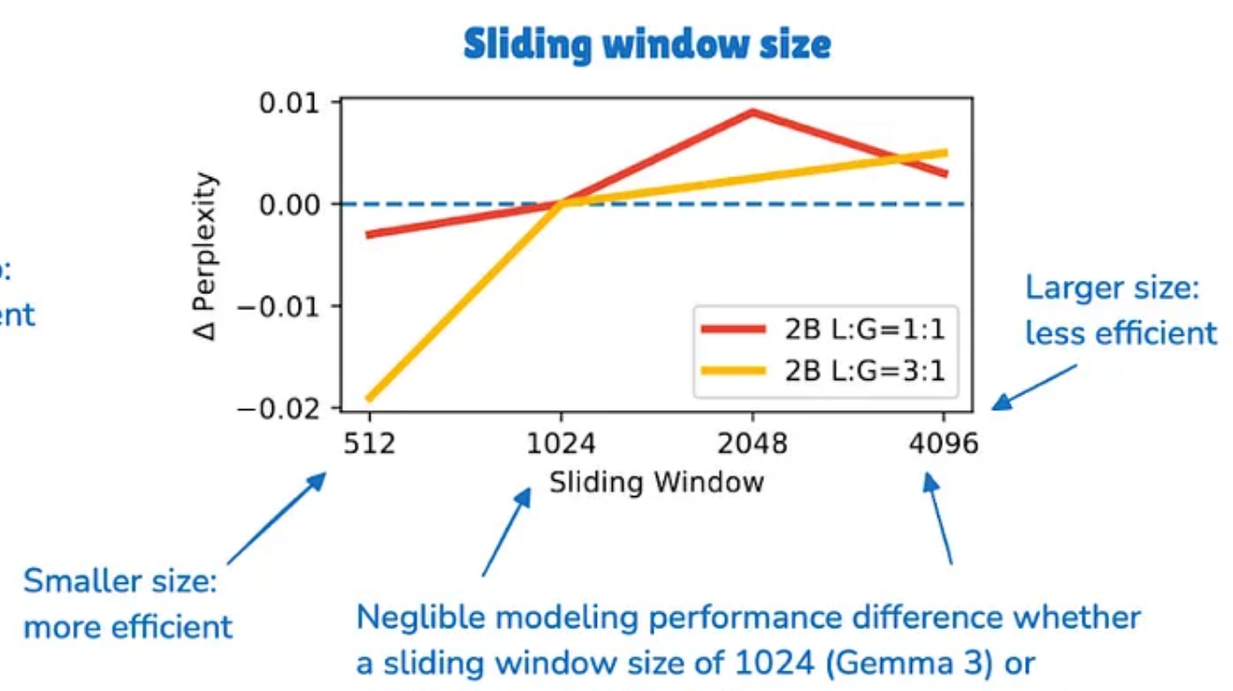
Sliding window size 4096

Gemma 3

1 full attention layer for every 5 layers using a sliding window 1:5

Sliding window size 1024





Post-Norm or Pre-Norm

Original transformer two normalization layers in the transformer block

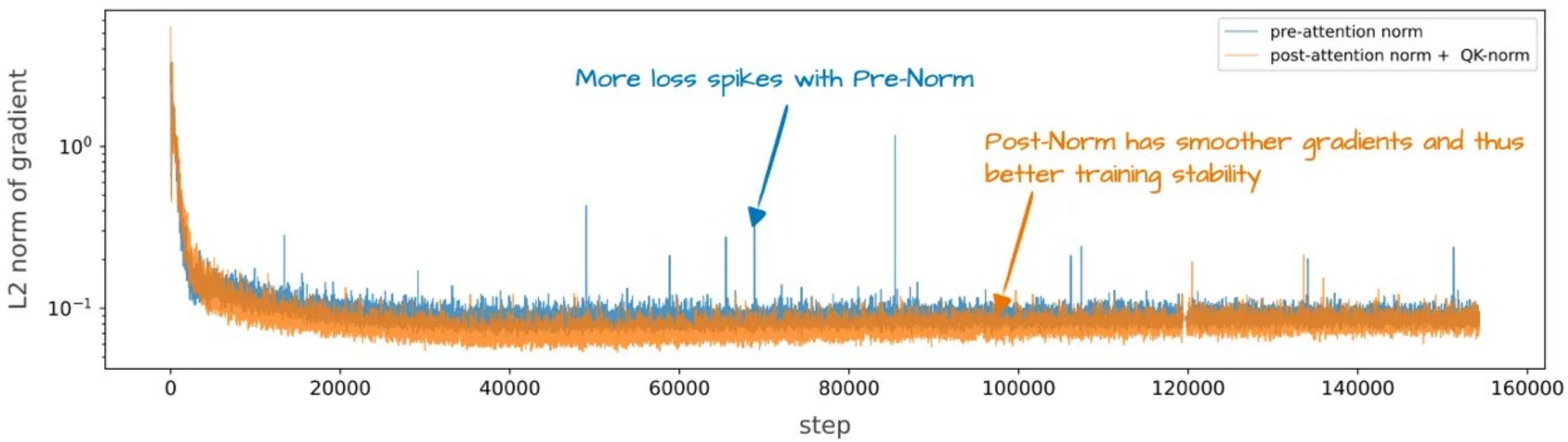
- After the attention module and the FeedForward module
- Post-Norm

GPT and most other LLMs

- Normalization layers before the attention and FeedForward modules
- Pre-Norm

Pre-Norm even works well without careful learning rate warm-up

With QK-Norm



QK-Norm

A RMSNorm layer - Root Mean Square Layer Normalization

Applied to queries and keys before applying RoPE

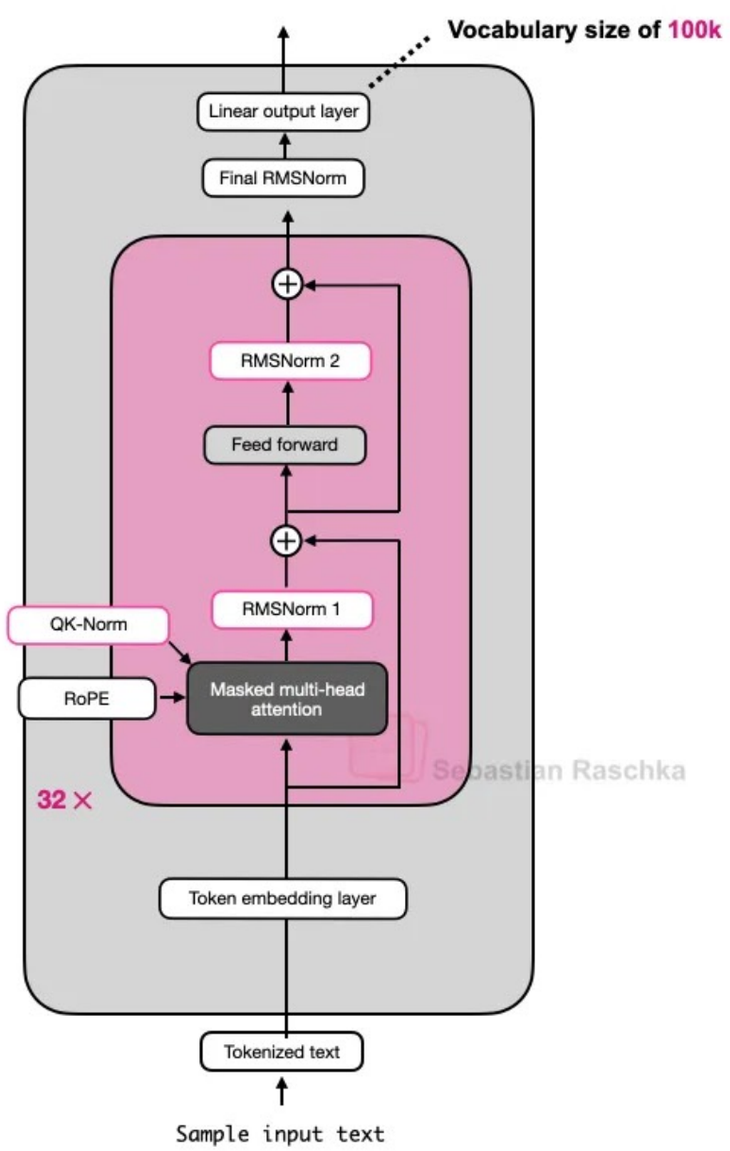
Normalization Layer Placement in Gemma 3

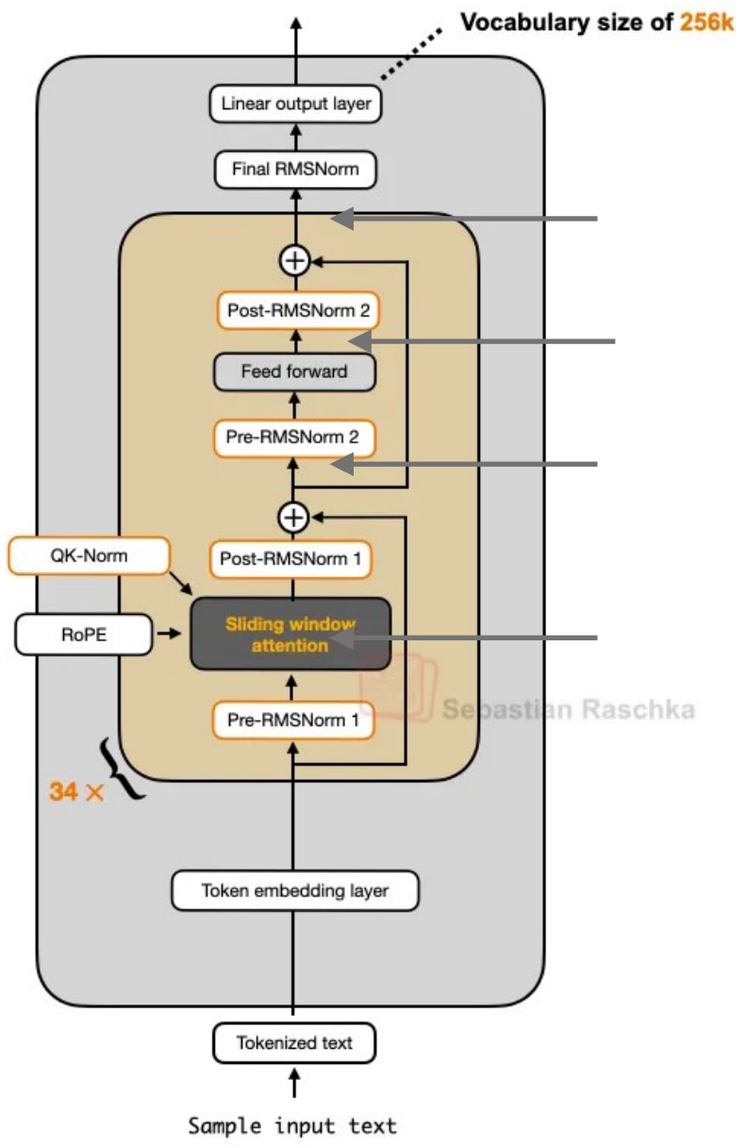
OLMo

- Post-Norm
- RMSNorm for queries & keys

Gemma 3

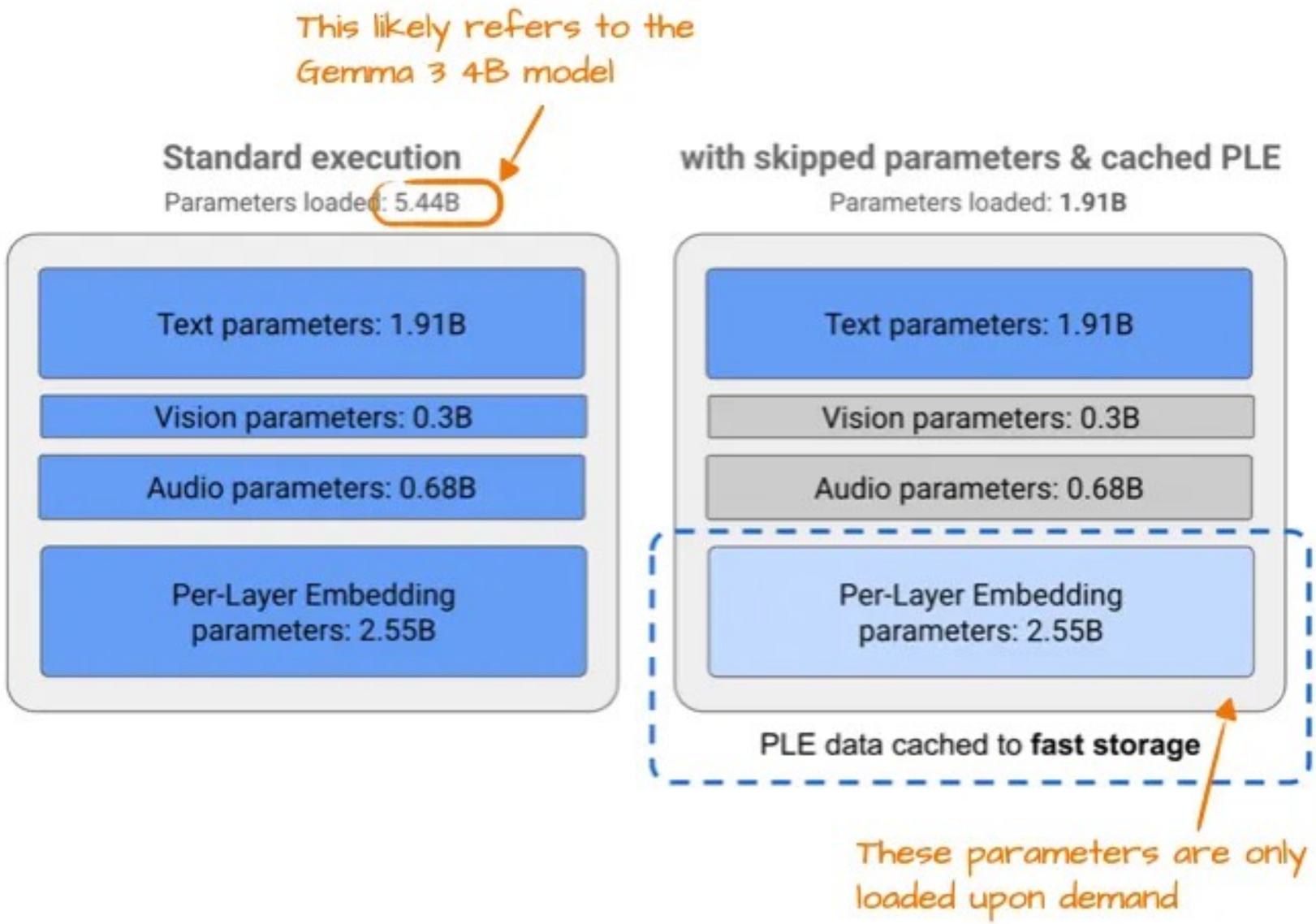
- Post & Pre-Norm





Gemma 3n

For small devices



Per-Layer Embedding (PLE)

Each layer has its own small parameter set that:

- Adjusts token representations
- Re-embeds or reprojects features for that specific layer

- Small linear layers or low-rank adapters
- Sometimes gating or scaling vectors

Each layer becomes specialized

Each layer has a look-up table for a token’s embedding for the layer

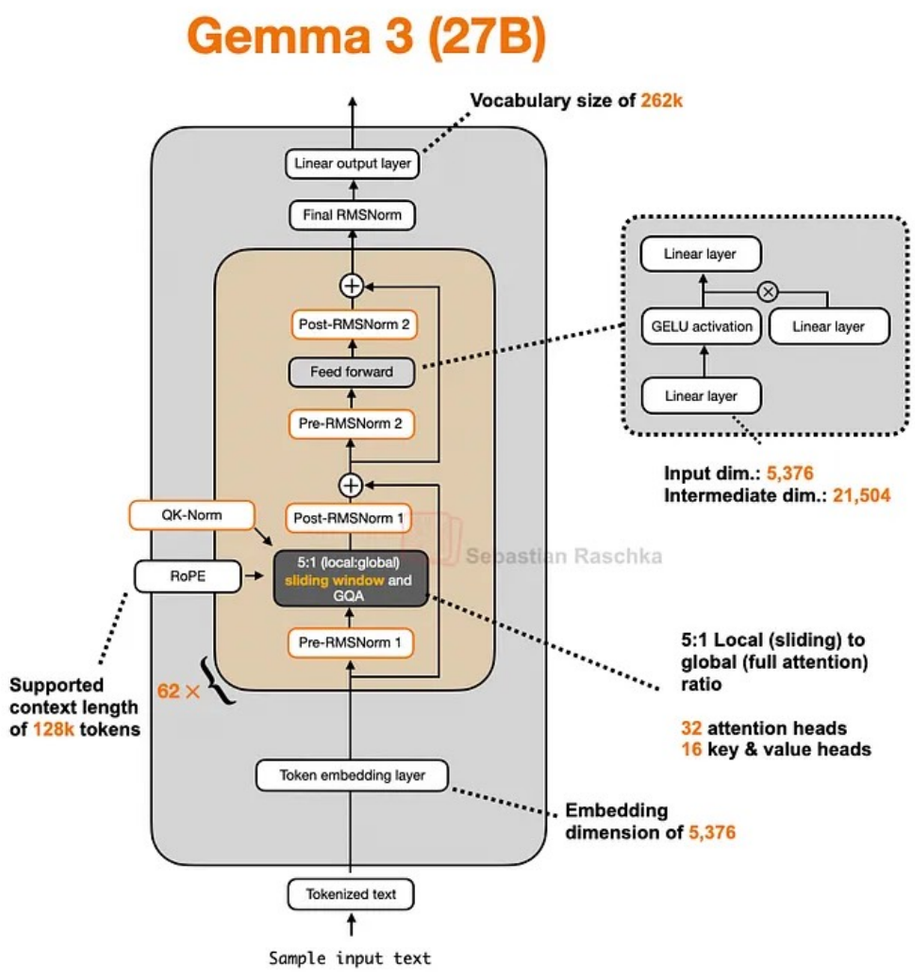
- Table is learned in training
- Fetched from main memory

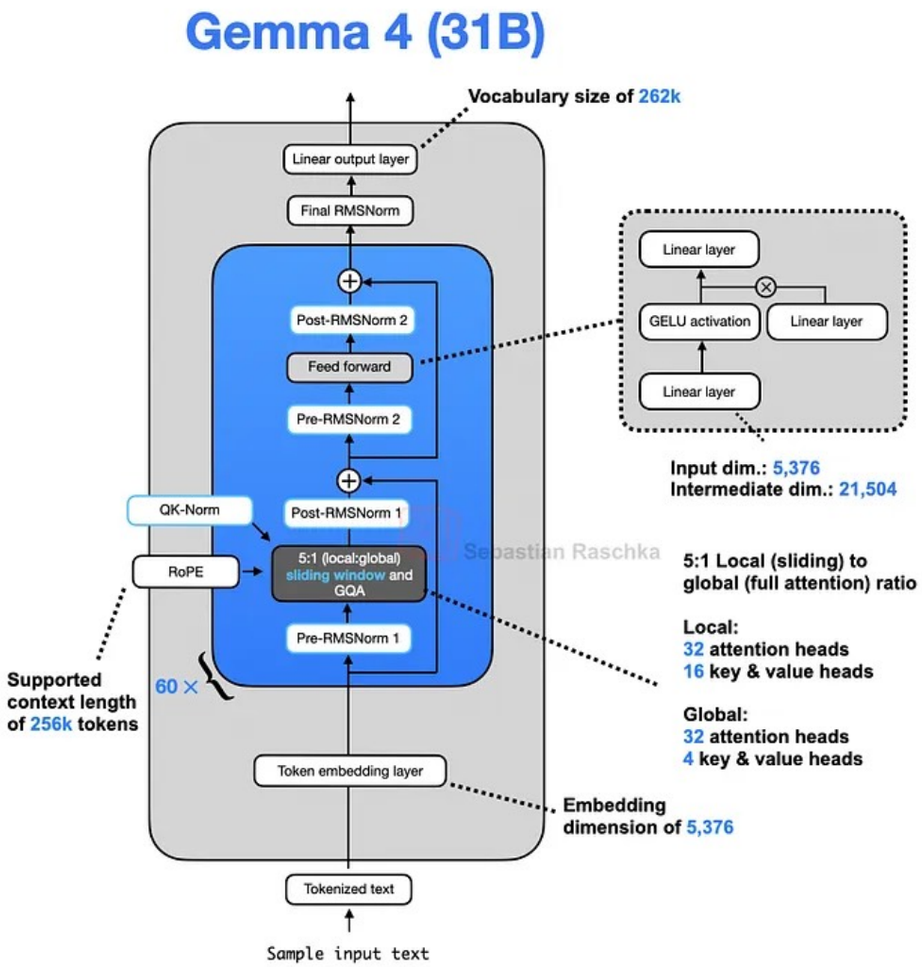
Gemma 4

Basically the same as Gemma 3

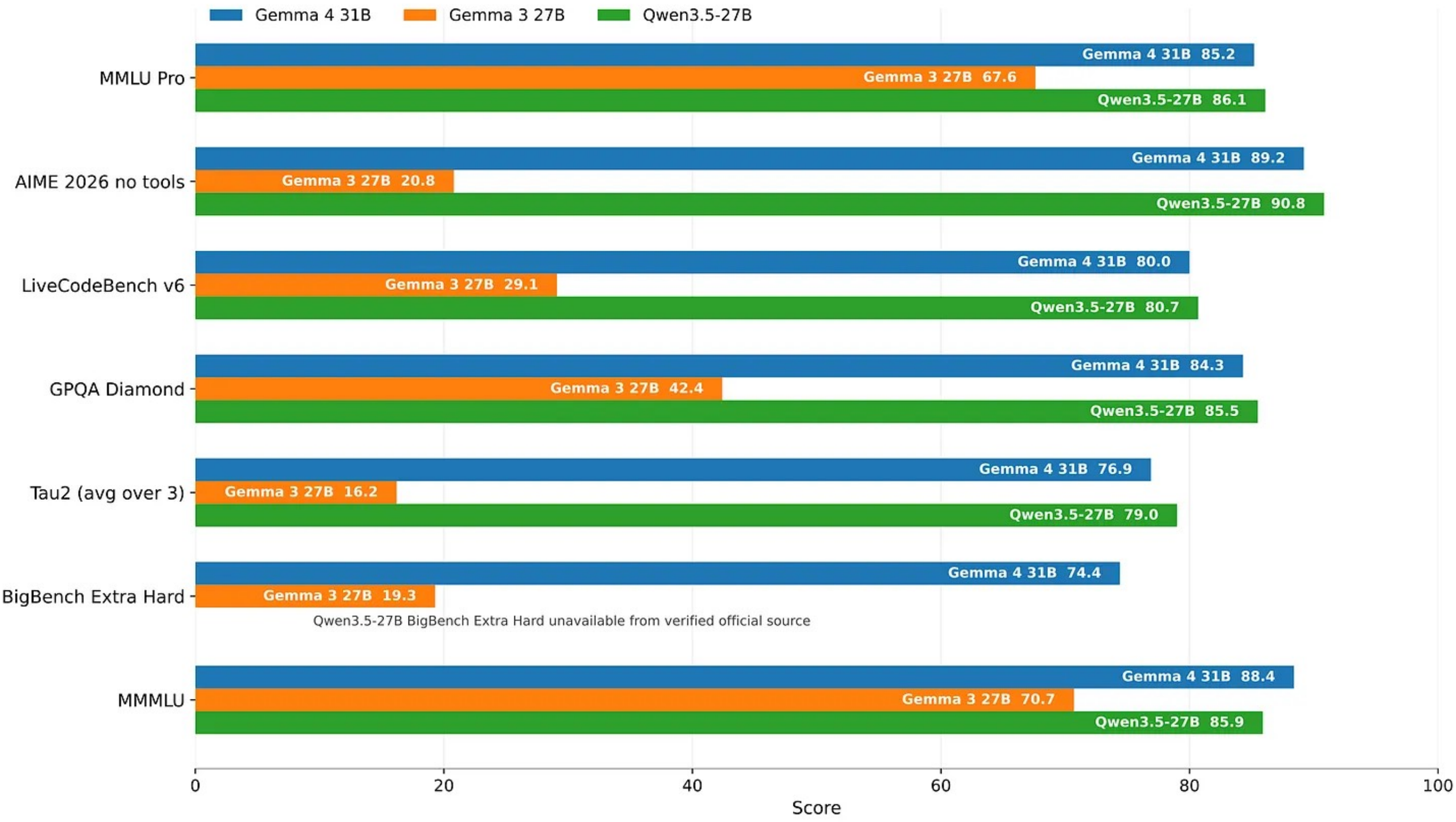
Gemma 4 Differences

- Uses p-RoPE
- For the full attention layers, they reuse the keys in the attention mechanism
  - Values = keys

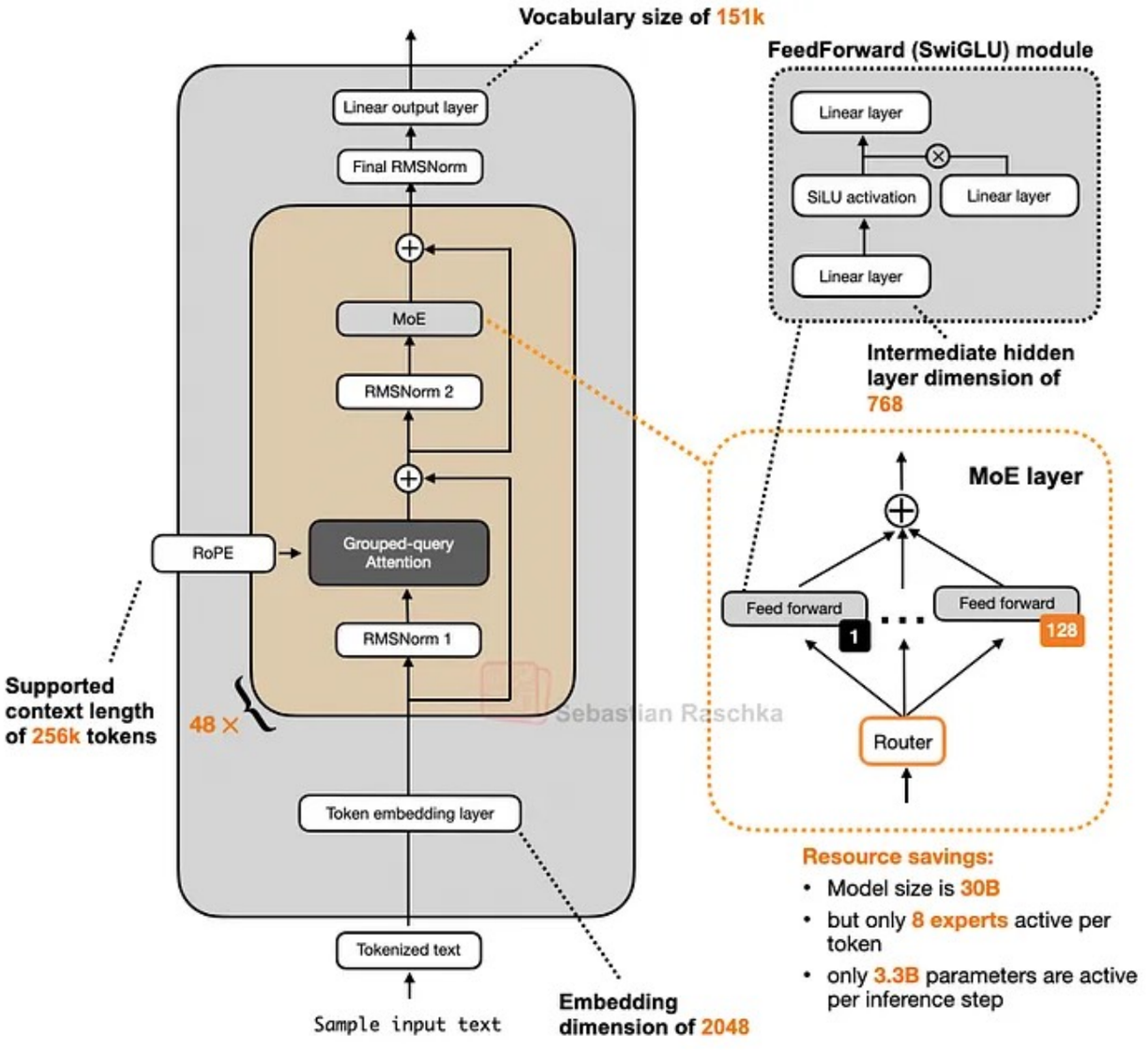


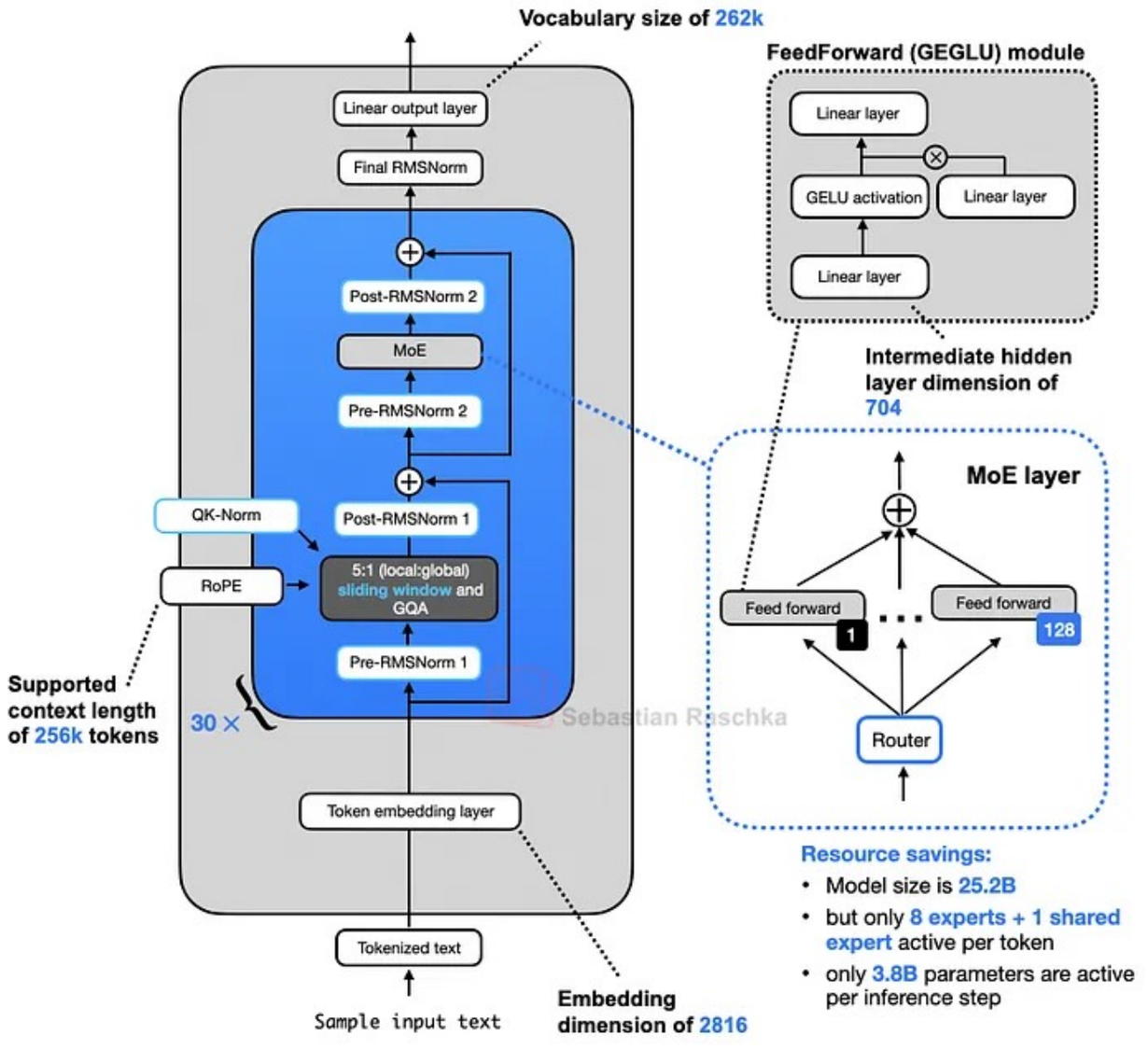


But Gema 4 is Better



Gemma 4 MoE 25B-A4B

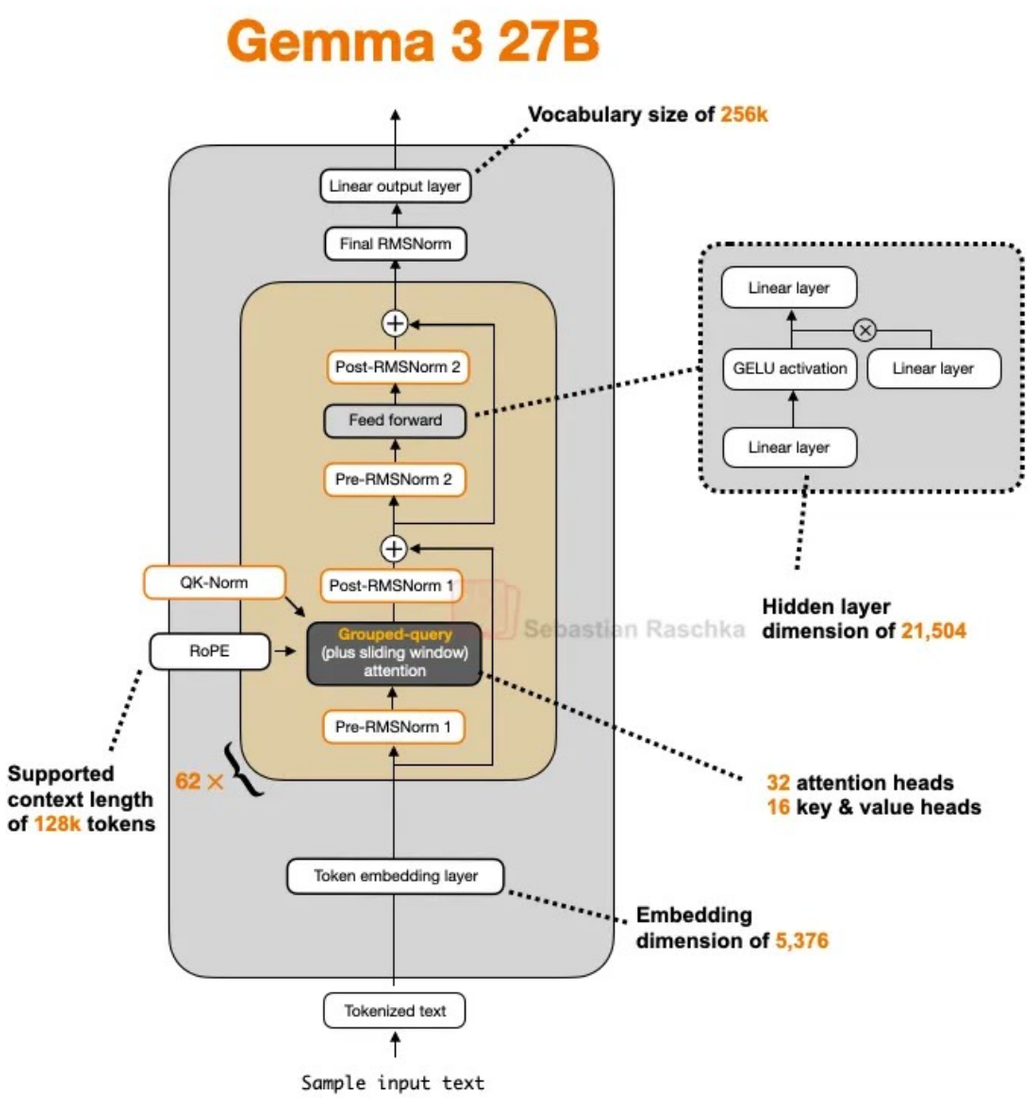


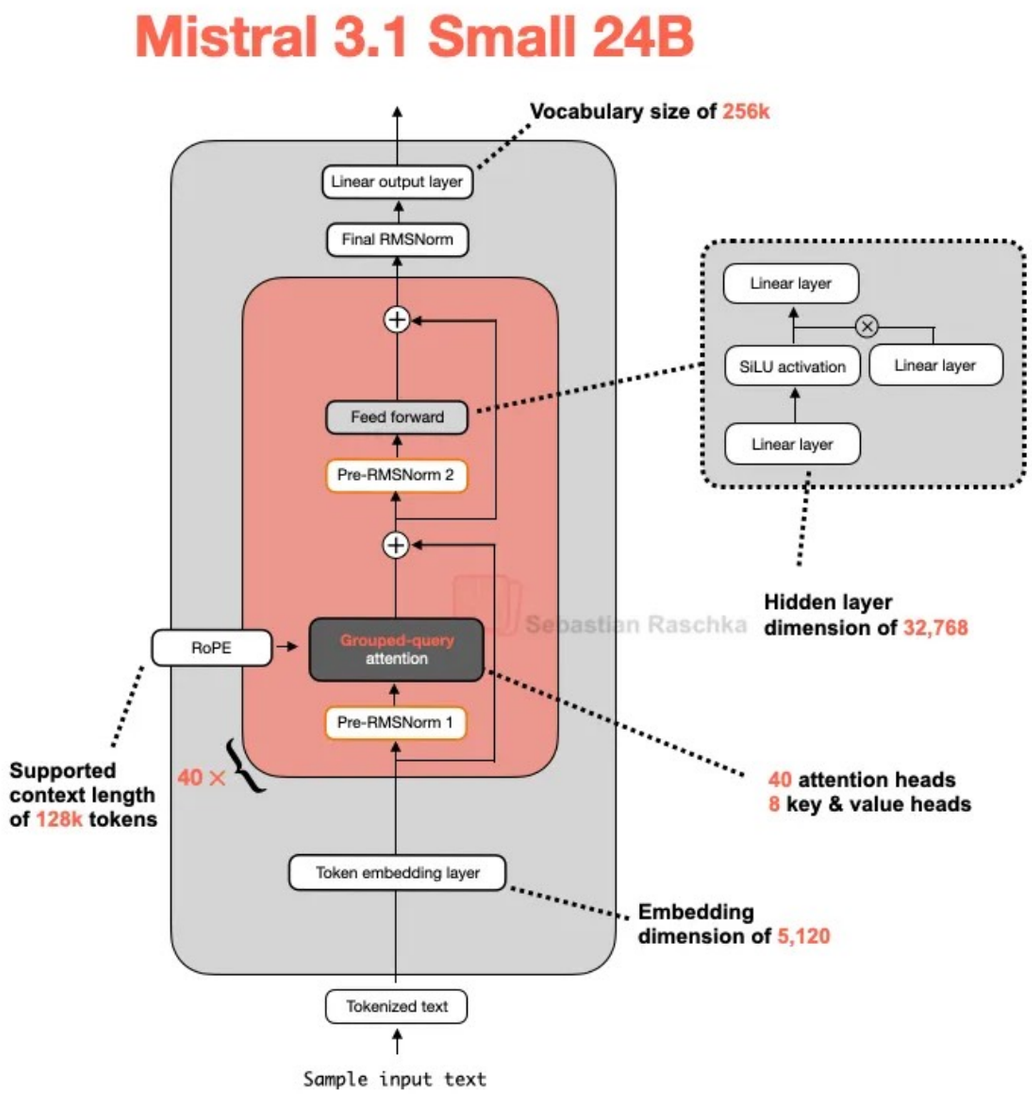


Mistral Small 3.1 24B

Outperforms Gemma 3 27B on several benchmarks while being faster

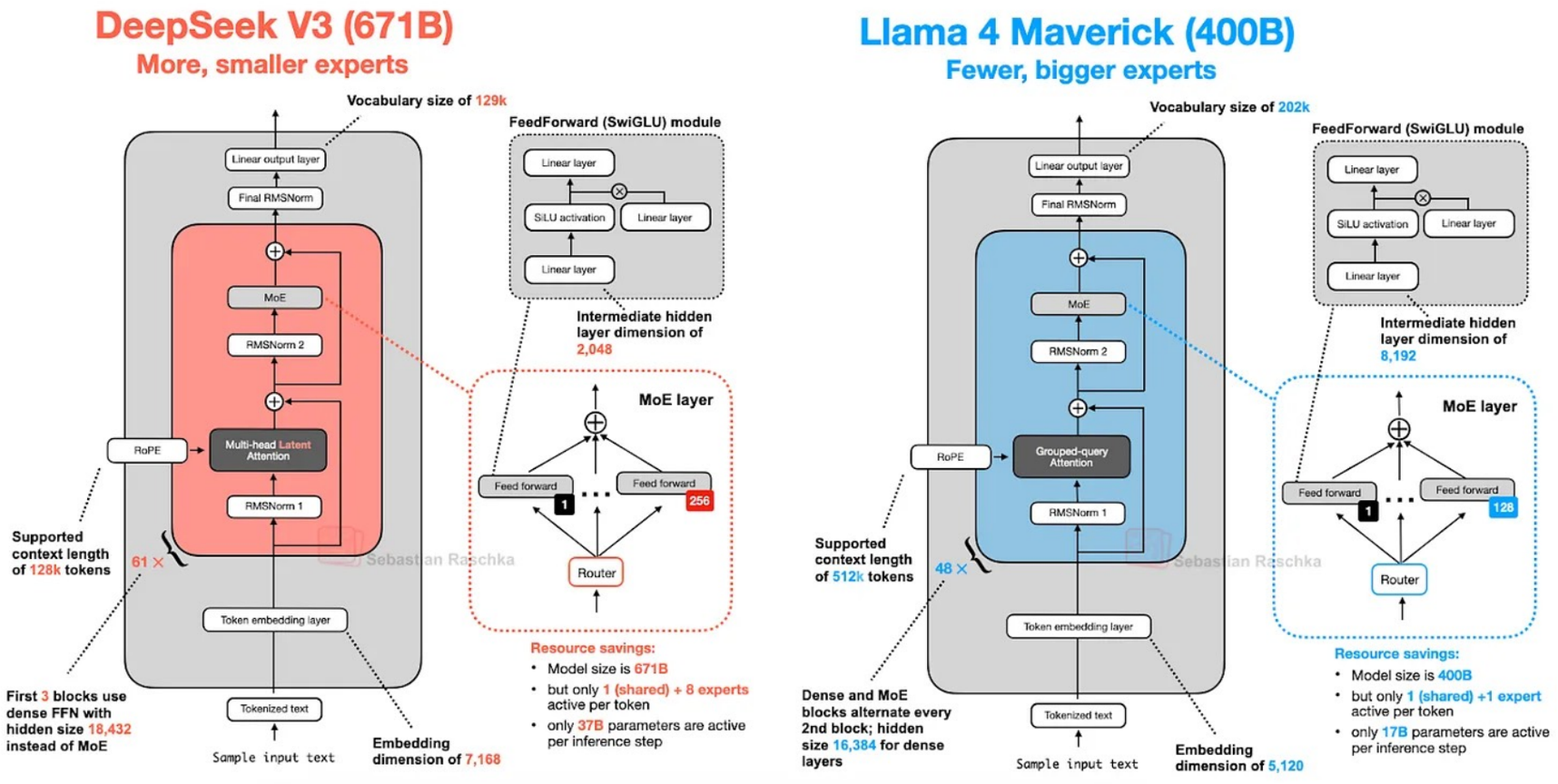
Custom tokenizer, and shrinking the KV cache and layer count





Llama 4 vs DeepSeek V3

DeepSeek V3	Llama 4
Multi-Head Latent Attention	Grouped-Query Attention
37 billion active parameters	17 billion active parameters
9 active experts	2 active experts
2,048 hidden size	8,192 hidden size
All layers but the first 3 are MoE	Alternates MoE & Dense layers

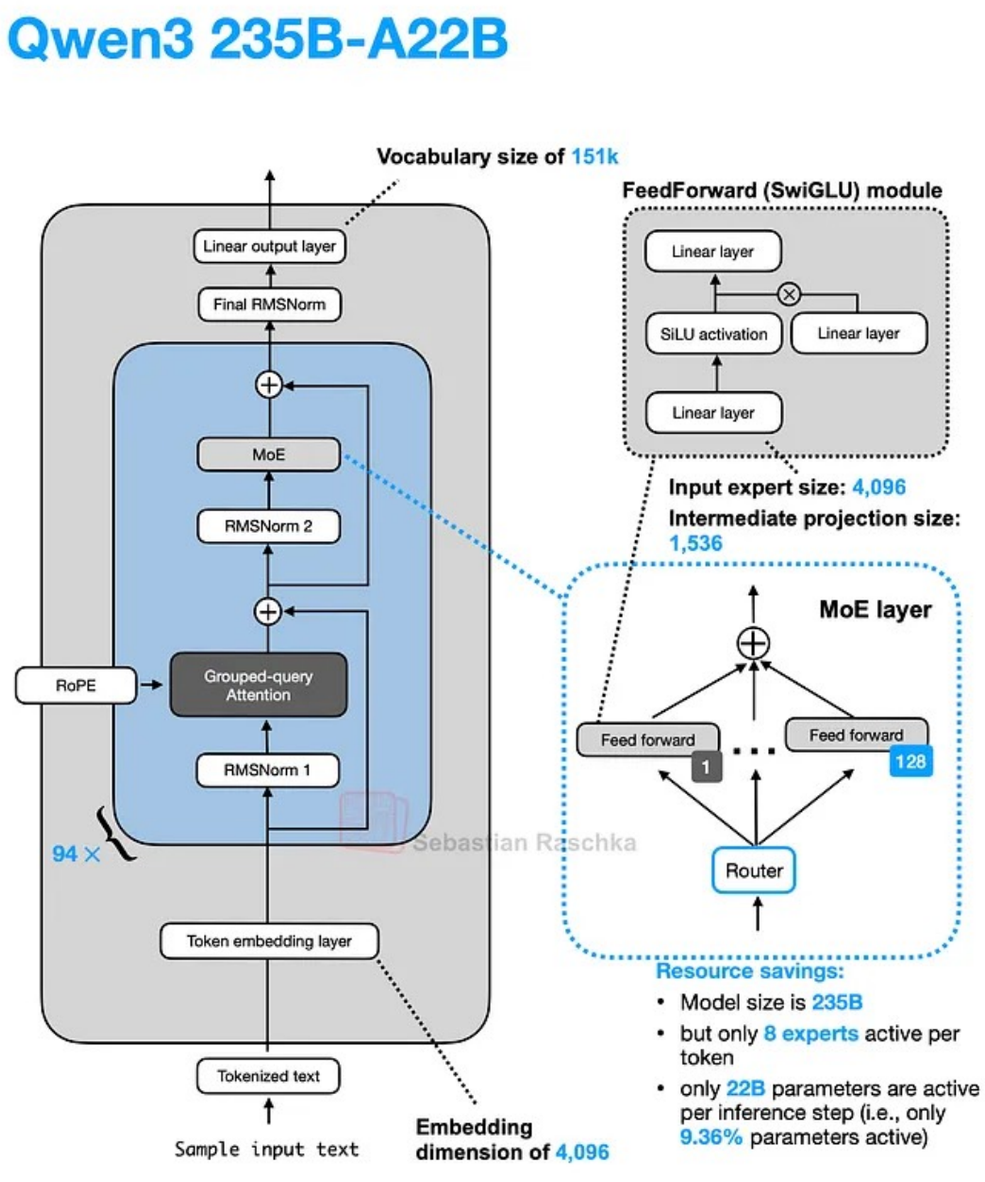


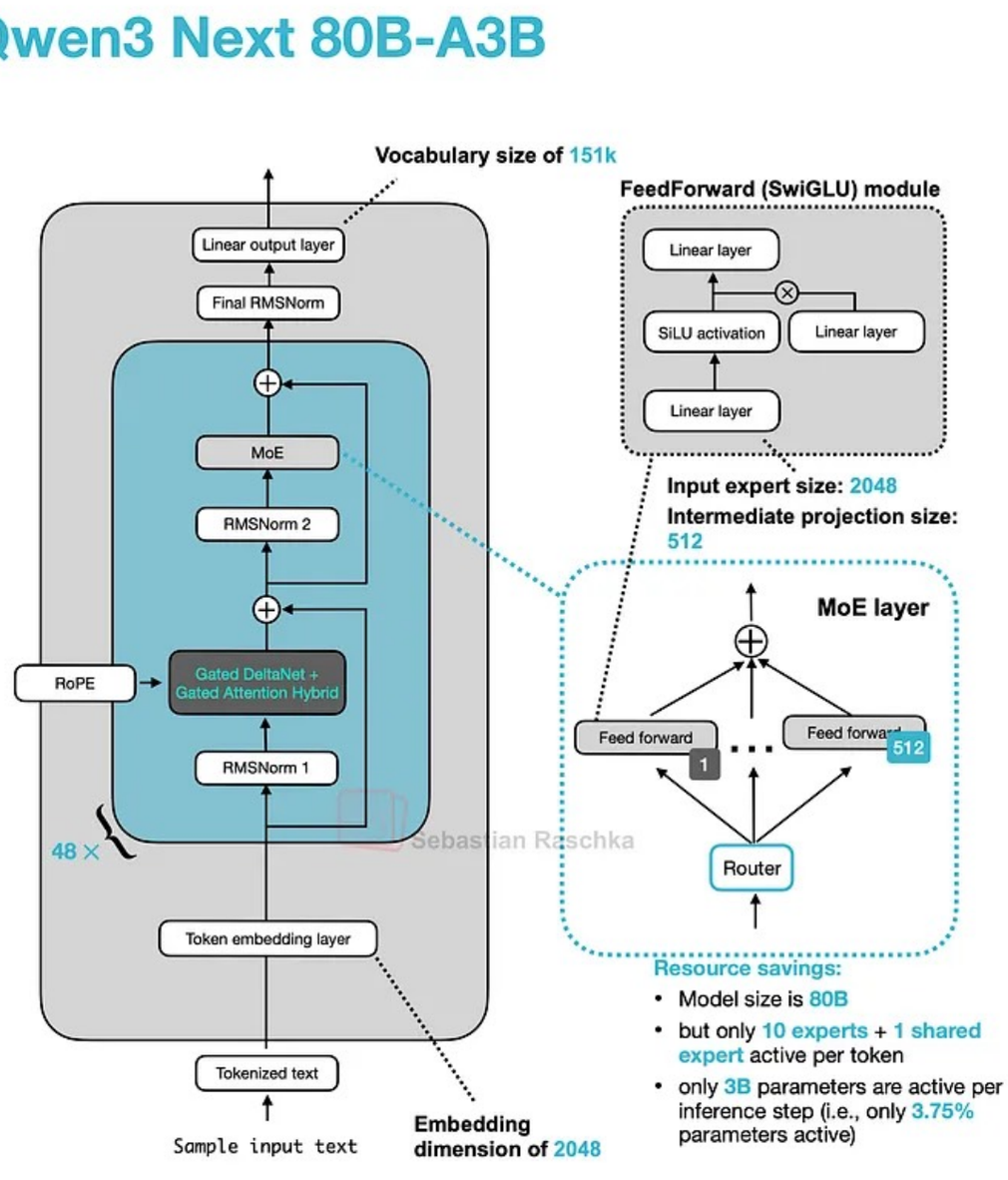
Qwen3-Next 80B-A3B

Qwen3 Next uses 4x more experts than Qwen3 235B-A22B

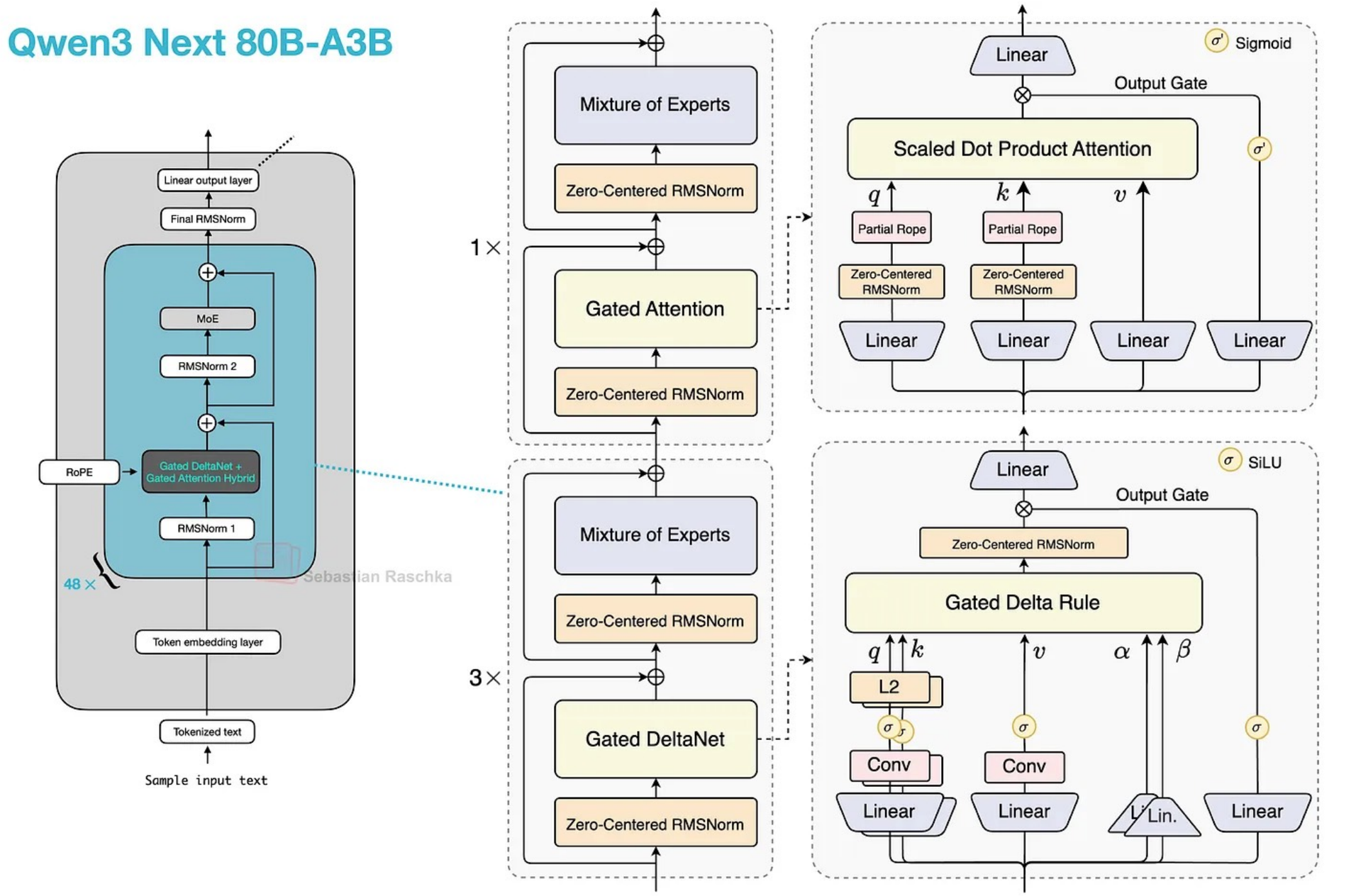
Qwen3 Next uses

- Gated DeltaNet + Gated Attention Hybrid
- Multi-Token Prediction (MTP)





Gated DeltaNet + Gated Attention Hybrid



Source: https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd&from=research.latest-advancements-list

3:1 ratio, 3 transformer blocks with Gated DeltaNet followed by 1 transformer block with Gated Attention

Gated Attention vs. Grouped-Query Attention (GQA)

Output gate that scales the attention result before it is added back to the residual

Zero-centered RMSNorm for QKNorm, rather than a standard RMSNorm

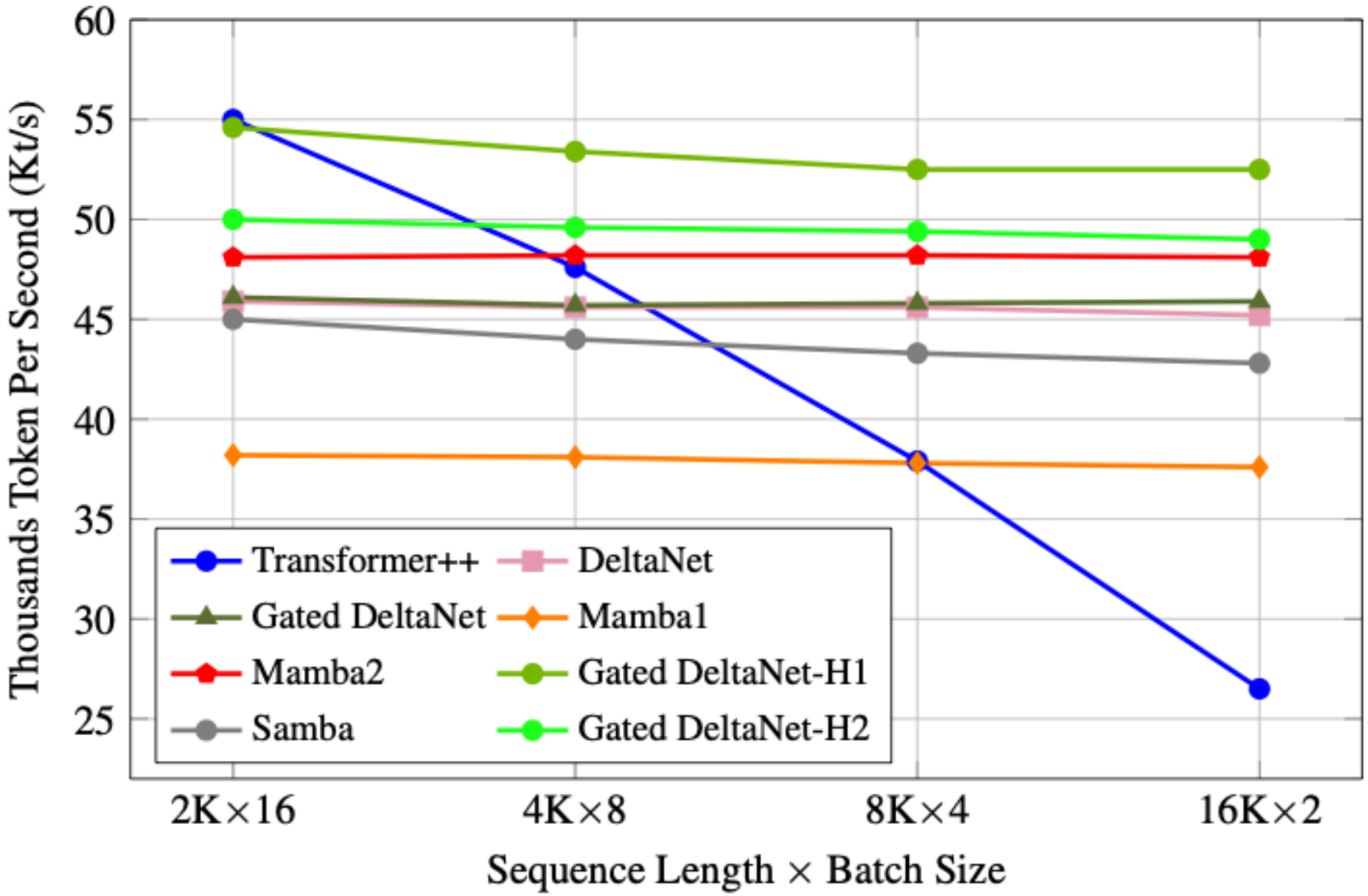
Partial RoPE on a subset of dimensions

Stability changes to GQA

Gated DeltaNet

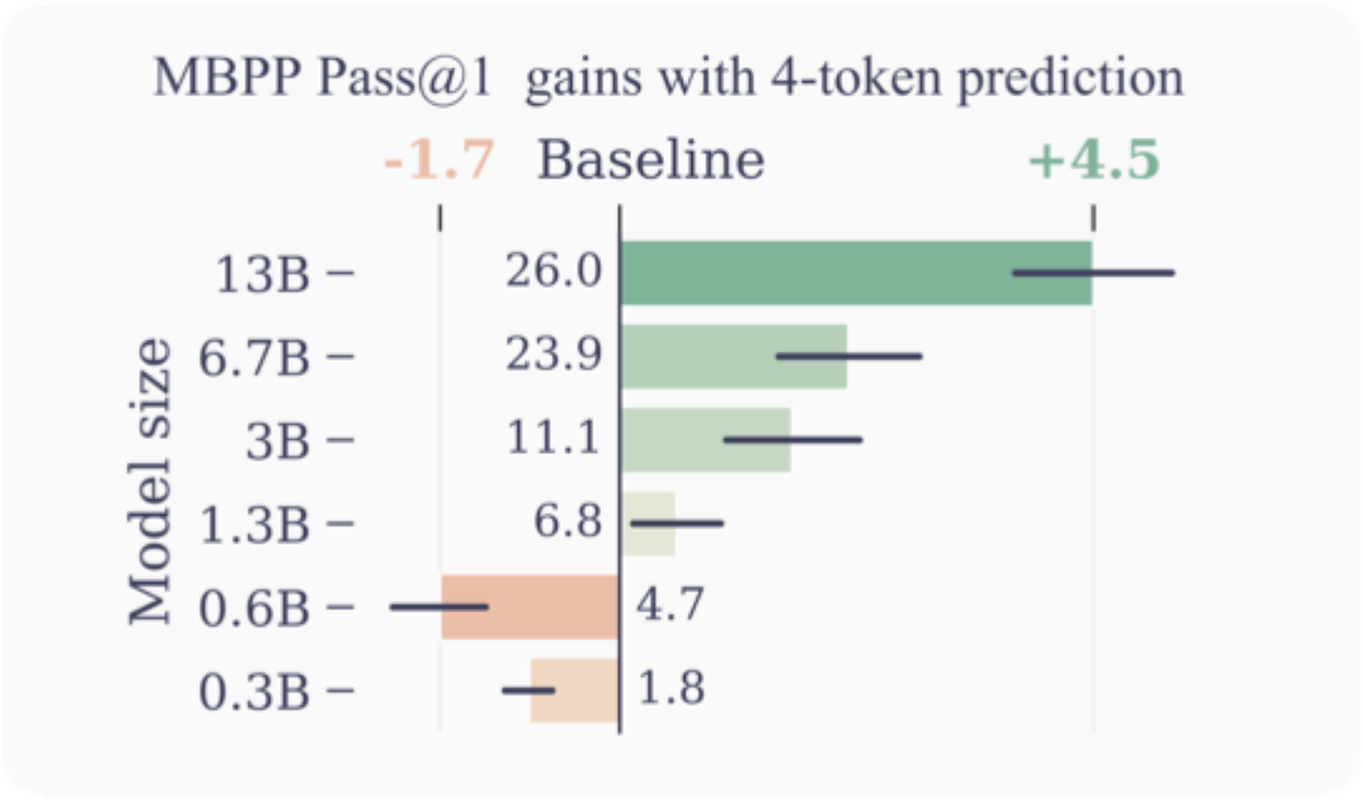
Reduces memory size

Less precise content retrieval, so still need Gated Attention



Multi-Token Prediction (MTP)

Predict several future tokens, instead of a single one, at each step

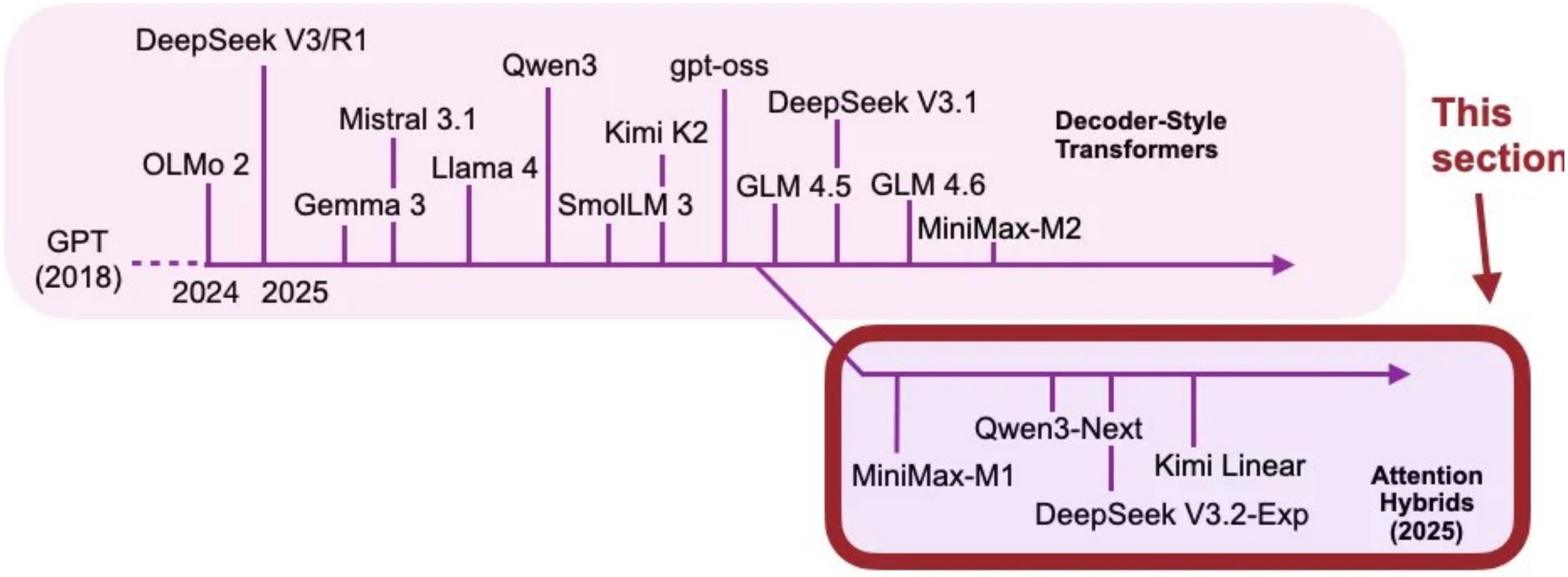


https://arxiv.org/pdf/2404.19737

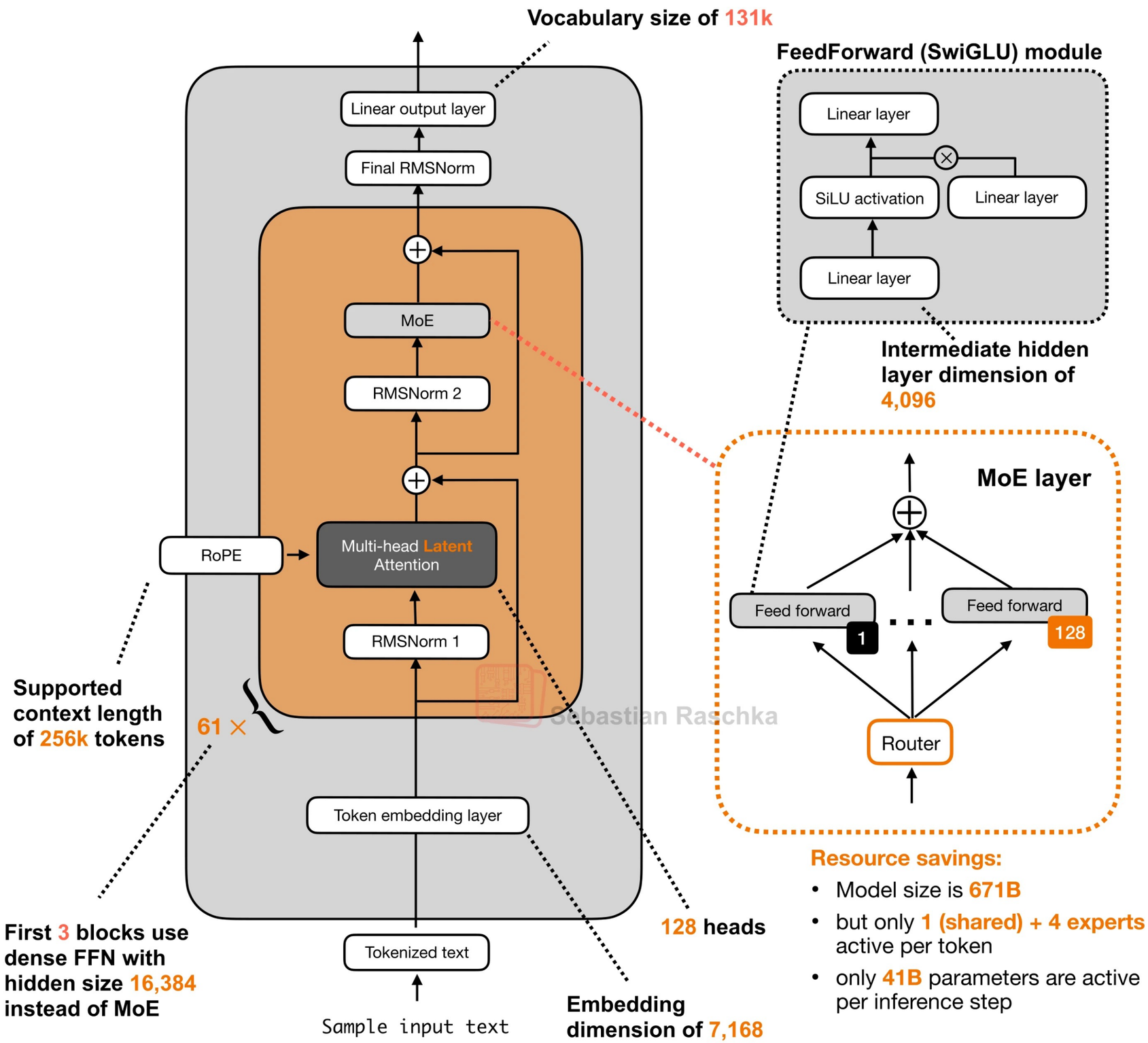
Better & Faster Large Language Models via Multi-token Prediction

30 Apr 2024

Linear Attention Revival



Mistral Large 3 (673B)



Resource savings:

- Model size is **671B**
- but only **1 (shared) + 4 experts** active per token
- only **41B** parameters are active per inference step