From Scratch to Scale:

Turning LLM Code into Architecture Insights

Sebastian Raschka (@rasbt)
https://sebastianraschka.com

PyCon DE &
PyData 2026

1

LLM ecosystem and tools

2

Understanding LLM architectures

3

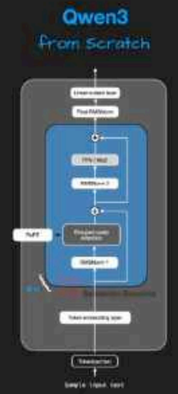
Trends & what's next for AI / LLMs

4

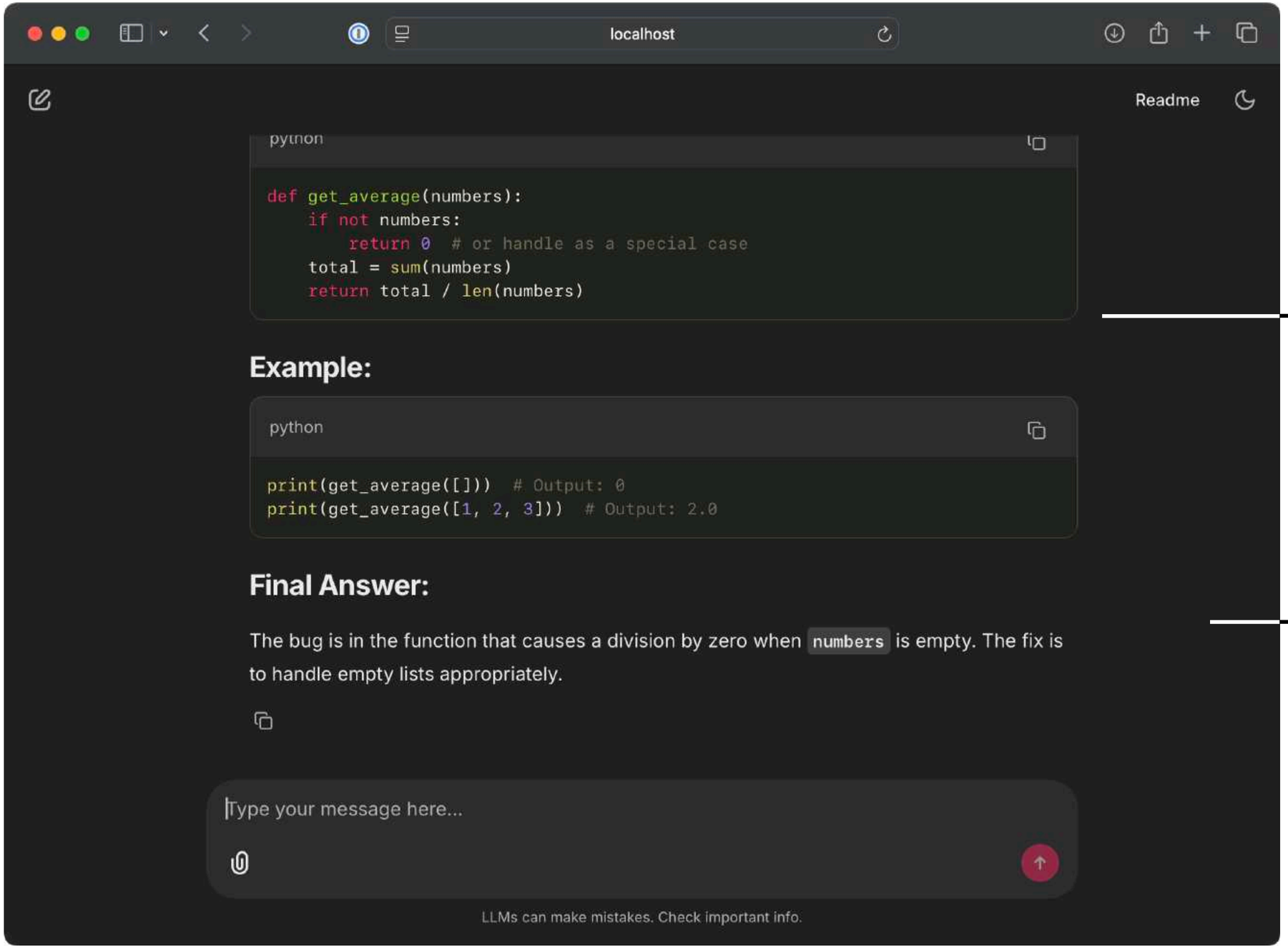
"Getting started" tips

localhost

Readme



Type your message here...



LLM

Languages

- Python 100.0%

Build A Reasoning Model (From Scratch)

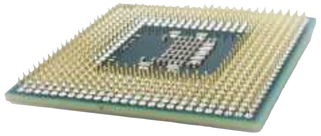
Interface

Languages

- Python 56.0%
- TypeScript 41.9%
- CSS 1.4%
- Other 0.7%

Chainlit

Python & PyTorch



CPU
`torch.device("cpu")`

ATen/C++ CPU kernels

Python code

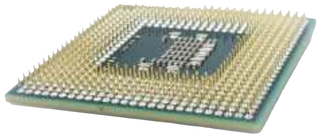
Lets us write code

PyTorch Python API/bindings

Dispatches operations

```
1  import torch
2  x = torch.tensor([1, 2, 3])
3  y = x * 2
```

Python & PyTorch



CPU
`torch.device("cpu")` → ATen/C++ CPU kernels

Python code → **PyTorch** Python API/bindings

Lets us write code

Dispatches operations

CUDA / ROCm
`torch.device("cuda")` → CUDA kernel for NVIDIA GPUs
ROCm for AMD GPUs

```
1  import torch
2  x = torch.tensor([1, 2, 3])
3  y = x * 2
```

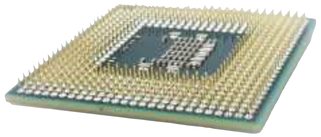
Python & PyTorch

Python code → **PyTorch** Python API/bindings

Lets us write code

Dispatches operations

```
1  import torch
2  x = torch.tensor([1, 2, 3])
3  y = x * 2
```



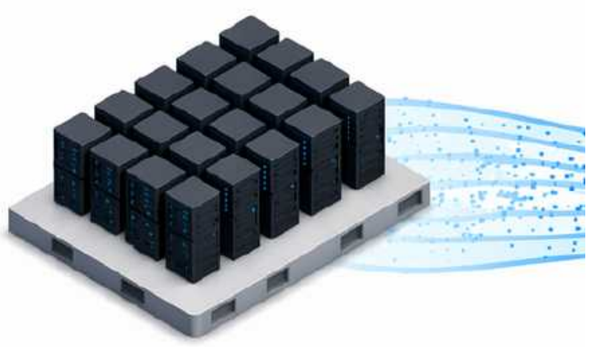
CPU
`torch.device("cpu")` → ATen/C++ CPU kernels

CUDA / ROCm
`torch.device("cuda")` → CUDA kernel for NVIDIA GPUs
ROCm for AMD GPUs

MPS
`torch.device("mps")` → Apple's Metal backend

LLM ecosystem and tools

Training



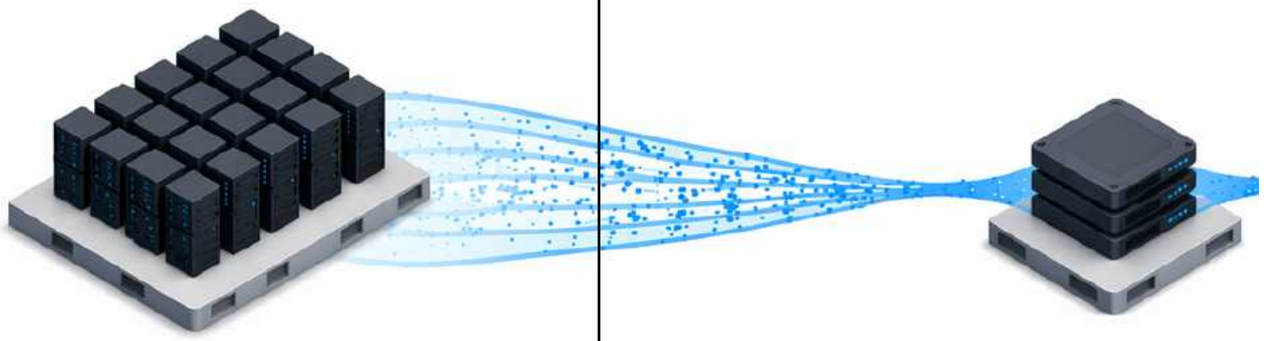
Python

PyTorch
Jax

CUDA

Training

Pre-trained weights



Python

PyTorch
Jax

CUDA

Hugging Face
model hub

~Everything goes through the model hub

Training

Pre-trained weights

Inference

Server-side

Python

PyTorch
Jax

CUDA

Hugging Face
model hub

SGLang

vllm

Training

Pre-trained weights

Inference

Python

PyTorch
Jax

CUDA

Hugging Face
model hub

Server-side

SGLang

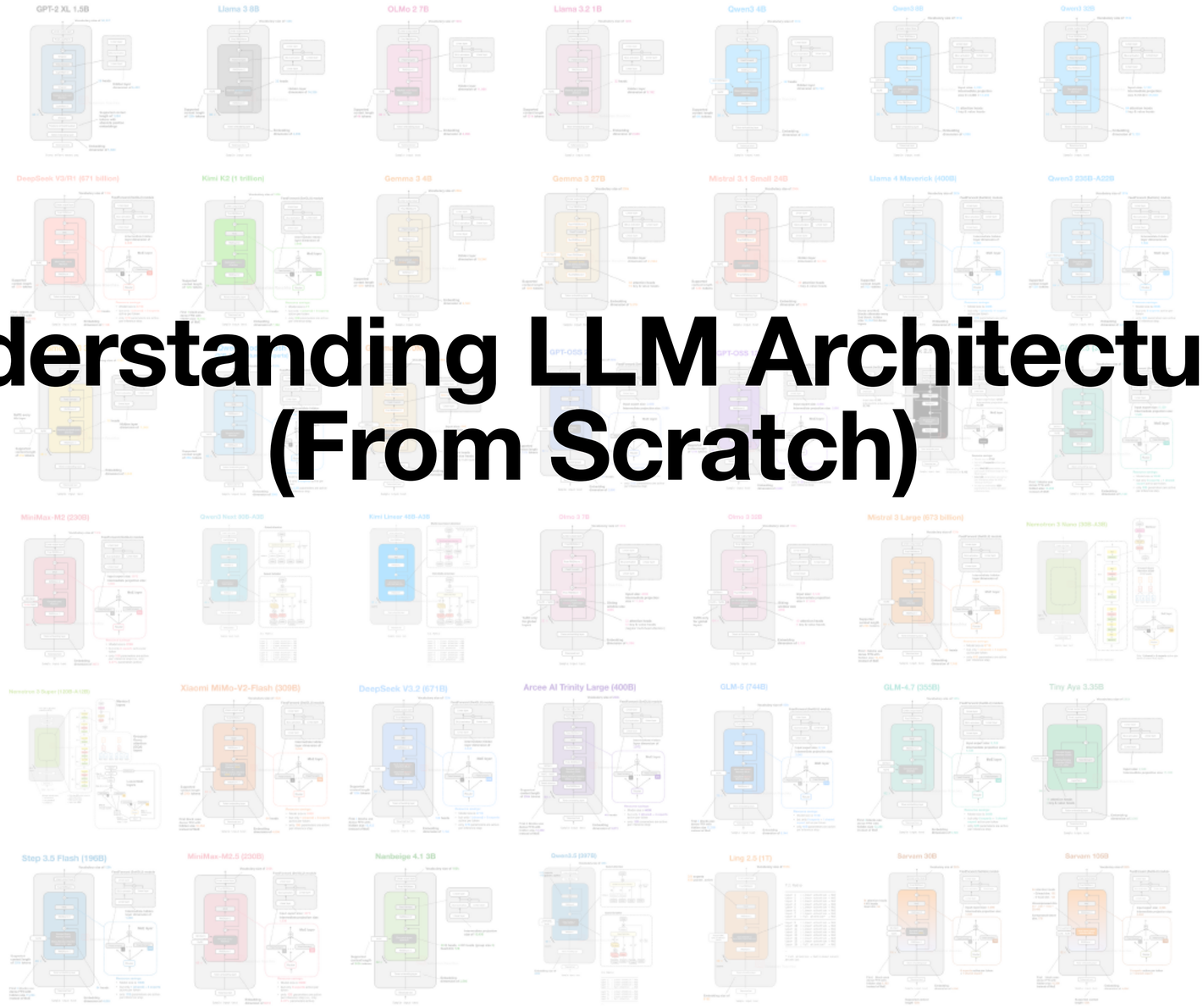
vllm

Local

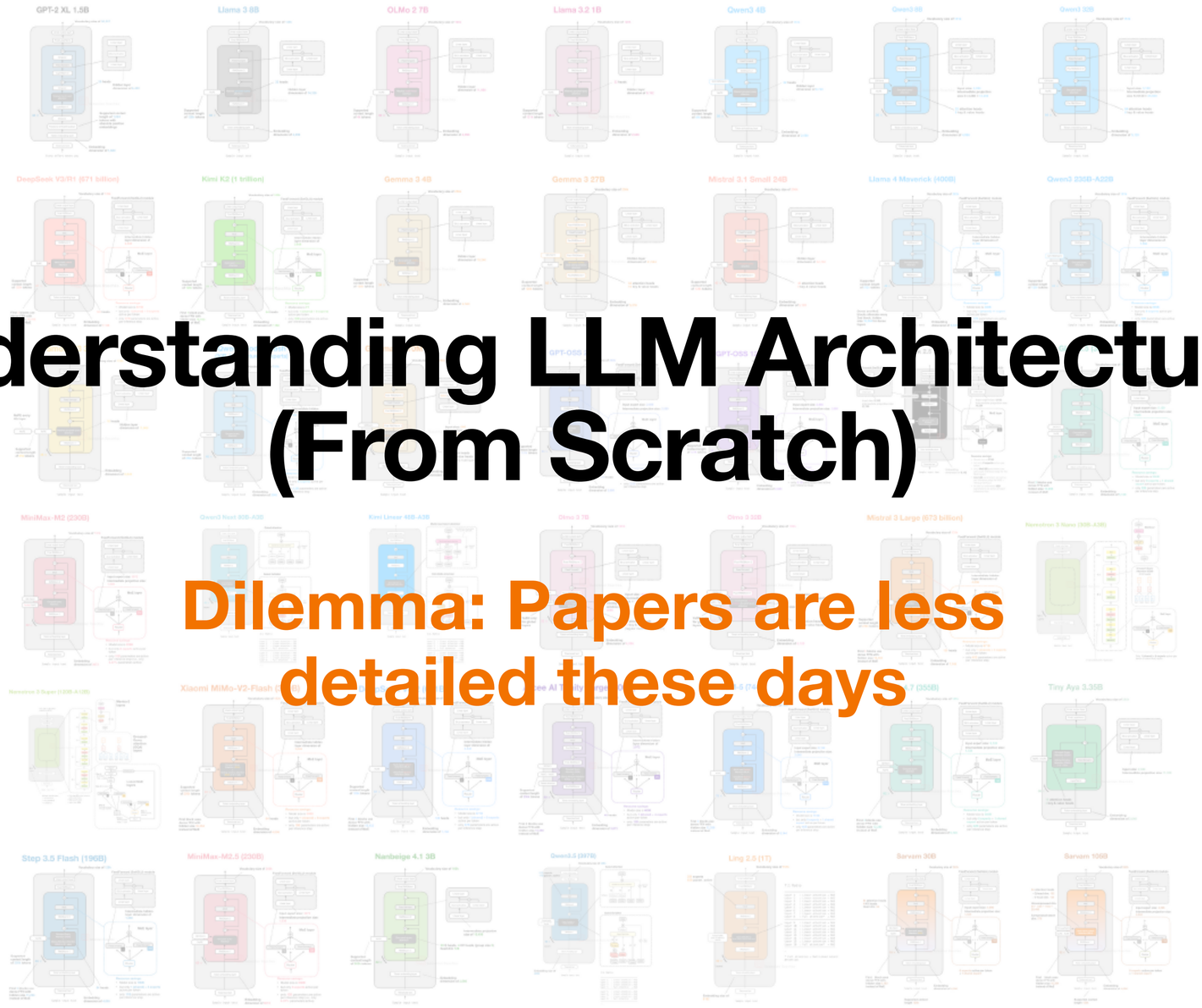
llama.cpp

MLX

Ollama



Understanding LLM Architectures (From Scratch)



Understanding LLM Architectures (From Scratch)

Dilemma: Papers are less detailed these days

But “working” code doesn’t lie

Understanding LLM Architectures (From Scratch)

From config files and code…

main gemma-3-270m / config.json

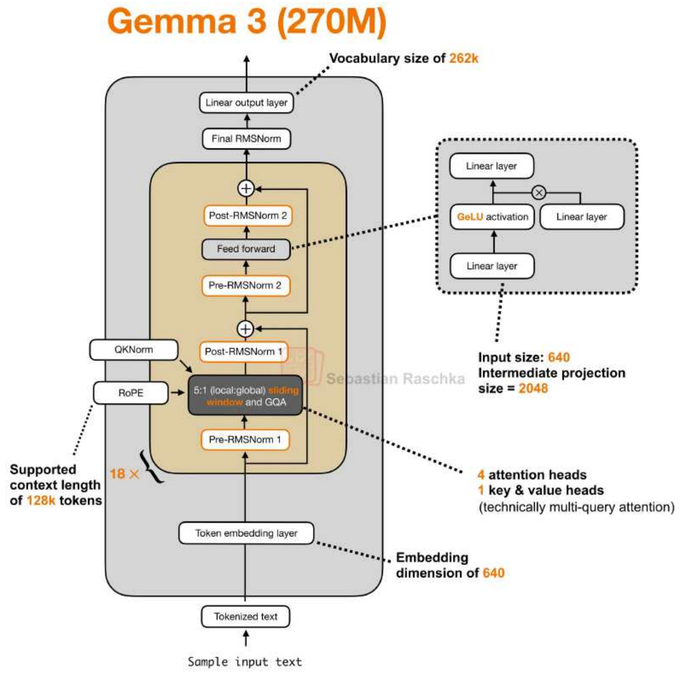
sindhuraghuram uploading the PT weights

raw Copy download link history blame

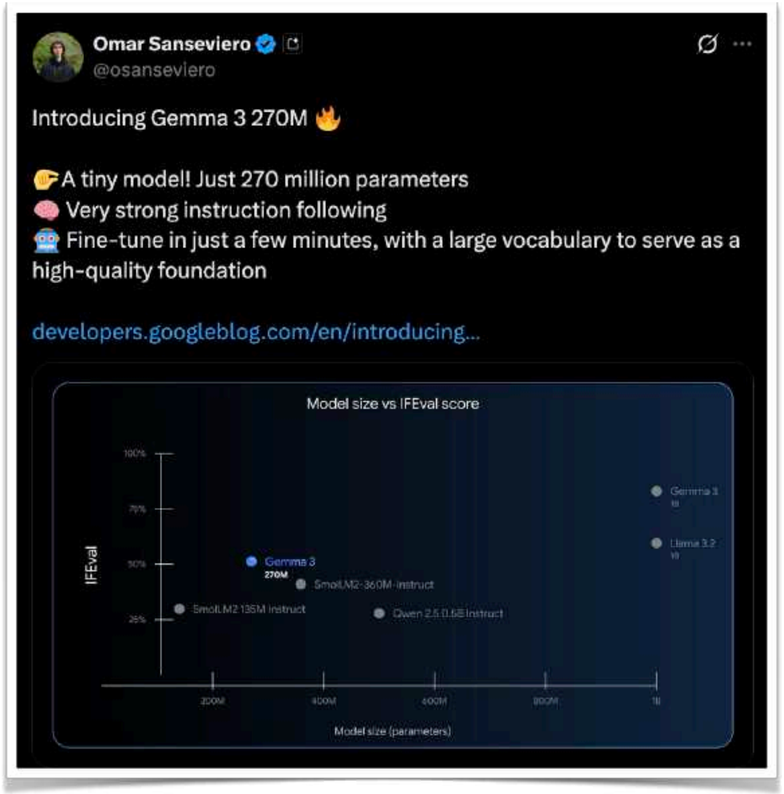
```json
{
  "_sliding_window_pattern": 6,
  "architectures": [
    "Gemma3ForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "attn_logit_softcapping": null,
  "bos_token_id": 2,
  "eos_token_id": 1,
  "final_logit_softcapping": null,
  "head_dim": 256,
  "hidden_activation": "gelu_pytorch_tanh",
  "hidden_size": 640,
  "initializer_range": 0.02,
  "intermediate_size": 2048,
  "layer_types": [
    "sliding_attention",
    "sliding_attention",
```

```python
class GemmaRMSNorm(nn.Module):
    def __init__(self, dim: int, eps: float = 1e-6):
        super().__init__()
        self.eps = eps
        self.weight = nn.Parameter(torch.zeros(dim))

    def _norm(self, x):
        return x * torch.rsqrt(x.pow(2).mean(-1, keepdim=True) + self.eps)

    def forward(self, x):
        output = self._norm(x.float())
        # Llama does x.to(float16) * w whilst Gemma is (x * w).to(float16)
        # See https://github.com/huggingface/transformers/pull/29402
        output = output * (1.0 + self.weight.float())
        return output.type_as(x)

    def extra_repr(self):
        return f"{tuple(self.weight.shape)}, eps={self.eps}"
```

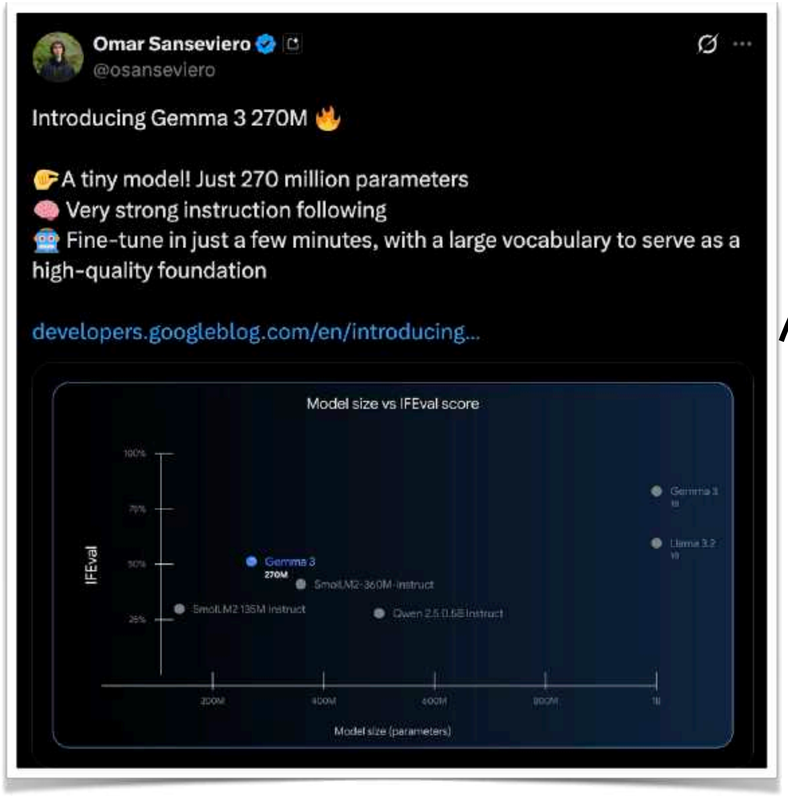
…to architecture insights

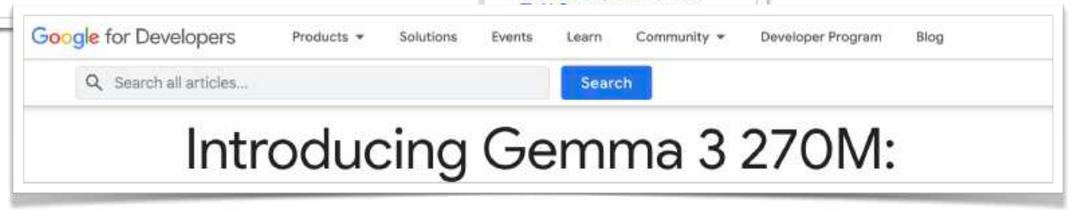


1 Follow the news



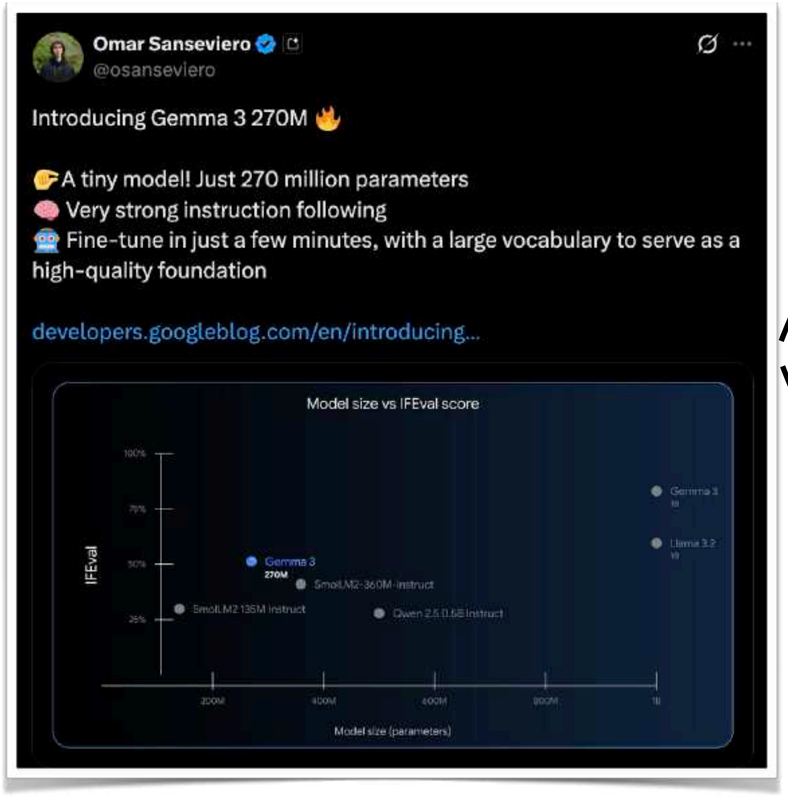
1 Follow the news

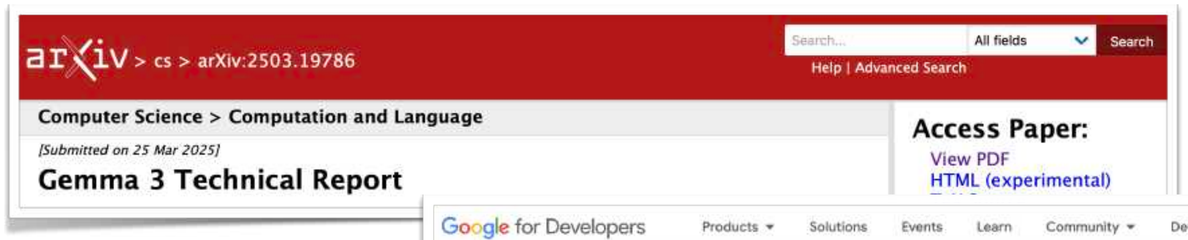


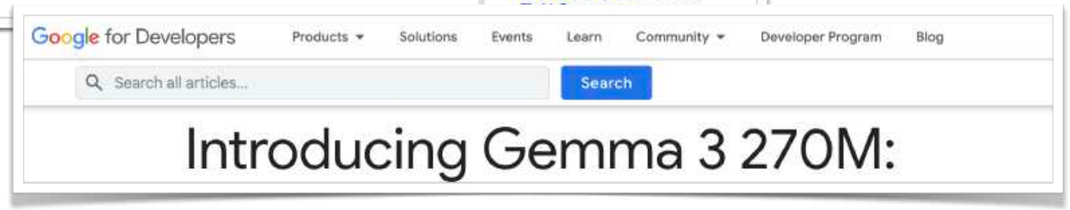


2 Read technical report(s)

1 Follow the news

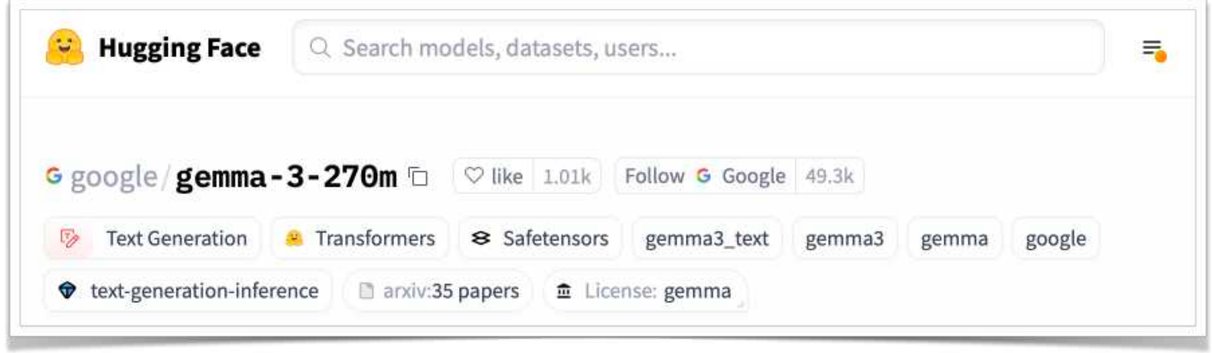




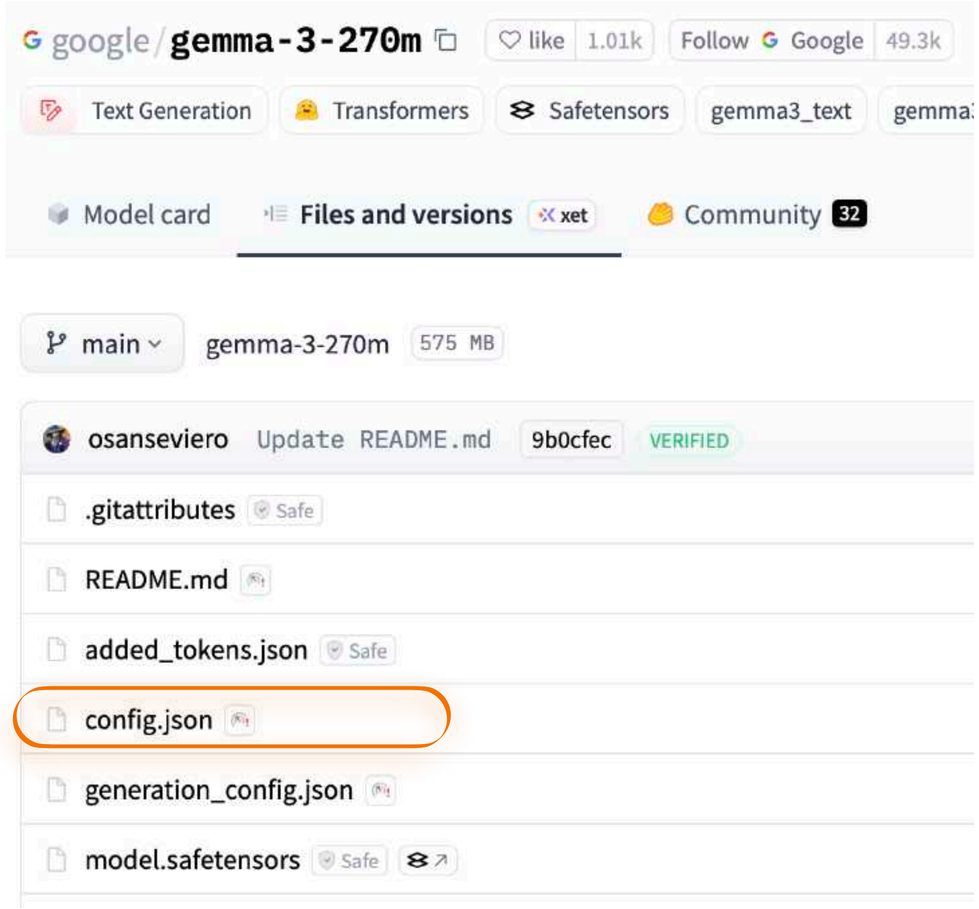


2 Read technical report(s)

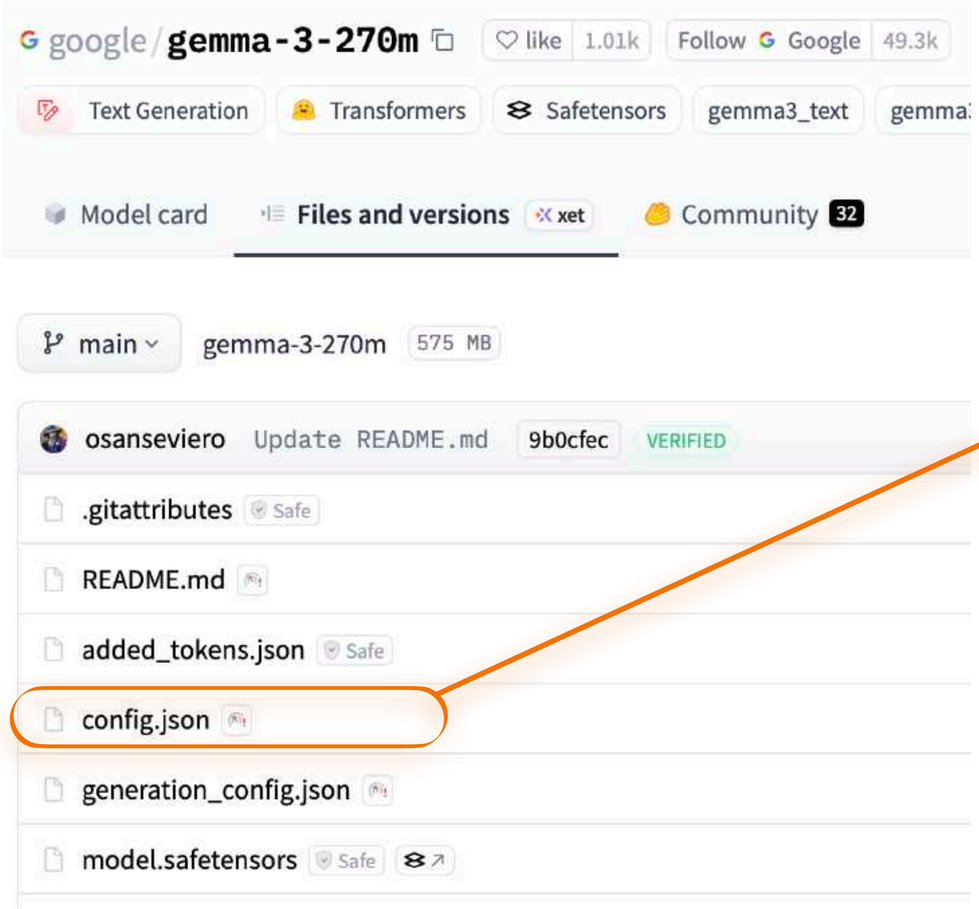
3 Check model hub page



4 Check `config.json`



4 Check `config.json`



main gemma-3-270m / config.json

sindhuraghuram uploading the PT weights for Gemma 3 270m 603ad58

raw Copy download link history blame contribute delete


```
{
  "_sliding_window_pattern": 6,
  "architectures": [
    "Gemma3ForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "attn_logit_softcapping": null,
  "bos_token_id": 2,
  "eos_token_id": 1,
  "final_logit_softcapping": null,
  "head_dim": 256,
  "hidden_activation": "gelu_pytorch_tanh",
  "hidden_size": 640,
  "initializer_range": 0.02,
  "intermediate_size": 2048,
  "layer_types": [
    "sliding_attention",
    "sliding_attention",
    "sliding_attention",
    "sliding_attention",
    "sliding_attention",
    "full_attention",
    "sliding_attention",
    "sliding_attention",
    "sliding_attention",
    "sliding_attention",
    "sliding_attention",
    "full_attention",
```

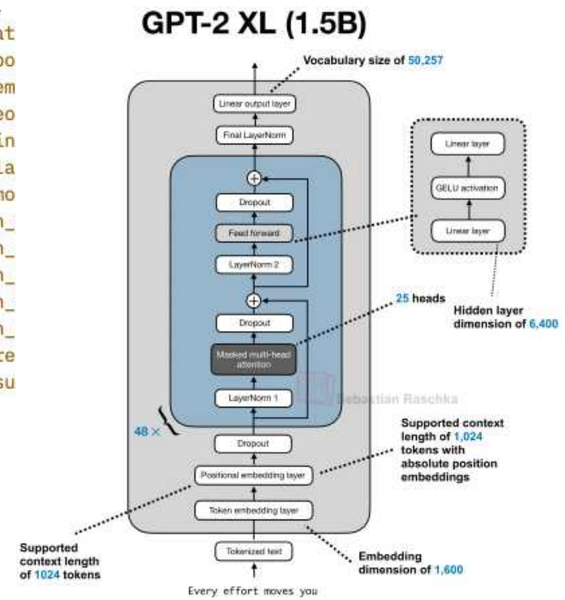
5 Compare with other architectures

main / gpt2 / config.json

system HF Staff Update config.json 7979bb7

raw Copy download link history blame contribute

```
{
  "activation_function": "gelu_new",
  "architectures": [
    "GPT2LMHeadModel"
  ],
  "at
  "bo
  "em
  "eo
  "in
  "la
  "mo
  "n_
  "n_
  "n_
  "n_
  "n_
  "re
  "su
```



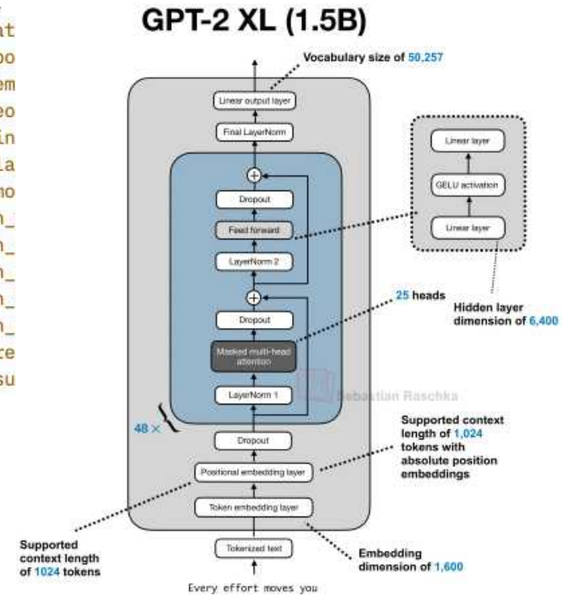
5 Compare with other architectures

main / gpt2 / config.json

system HF Staff Update config.json 7979bb7

raw | Copy download link | history | blame | contribute

```
{
  "activation_function": "gelu_new",
  "architectures": [
    "GPT2LMHeadModel"
  ],
  "at
  "bo
  "em
  "eo
  "in
  "la
  "mo
  "n_
  "n_
  "n_
  "n_
  "n_
  "re
  "su
```

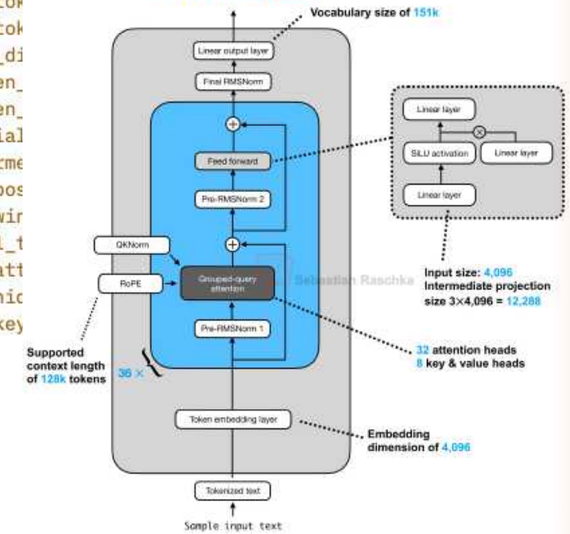


main / Qwen3-8B / config.json

littlebird13 Upload folder using huggingface_hub 47719a7

raw | Copy download link | history | blame | contribute

```
{
  "architectures": [
    "Qwen3ForCausalLM"
  ],
  "attention_bias": false,
  "attenti
  "bos_tok
  "eos_tok
  "head_di
  "hidden_
  "hidden_
  "initial
  "interme
  "max_pos
  "max_wir
  "model_t
  "num_att
  "num_hic
  "num_key
```



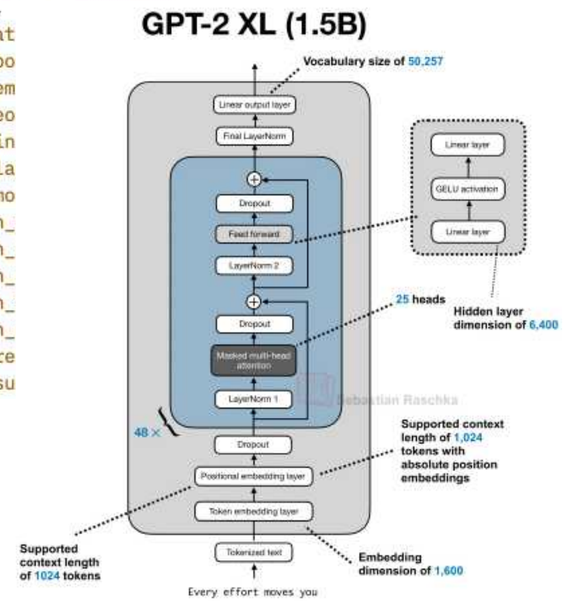
5 Compare with other architectures

main / gpt2 / config.json

system HF Staff Update config.json 7979bb7

raw | Copy download link | history | blame | contribute

```
{
  "activation_function": "gelu_new",
  "architectures": [
    "GPT2LMHeadModel"
  ],
  "at
  "bo
  "em
  "eo
  "in
  "la
  "mo
  "n_
  "n_
  "n_
  "n_
  "n_
  "re
  "su
```

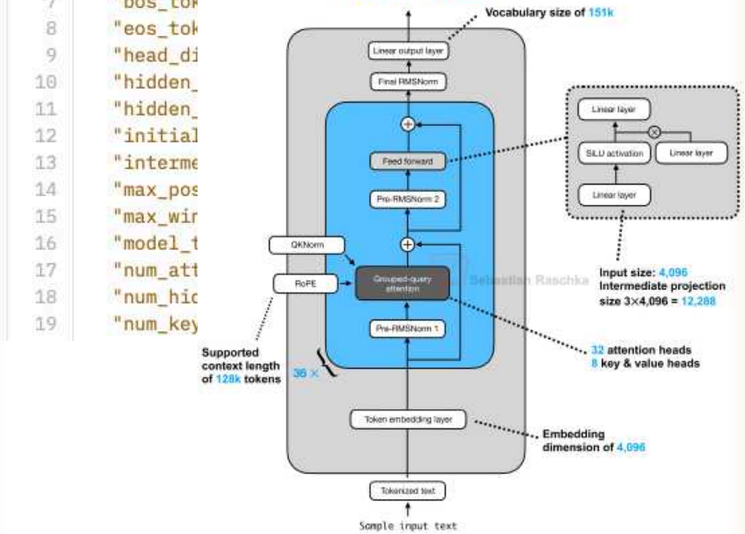


main / Qwen3-8B / config.json

littlebird13 Upload folder using huggingface_hub 47719a

raw | Copy download link | history | blame | contribute

```
{
  "architectures": [
    "Qwen3ForCausalLM"
  ],
  "attention_bias": false,
  "attenti
  "bos_tok
  "eos_tok
  "head_di
  "hidden_
  "hidden_
  "initial
  "interme
  "max_pos
  "max_wir
  "model_t
  "num_att
  "num_hic
  "num_key
```

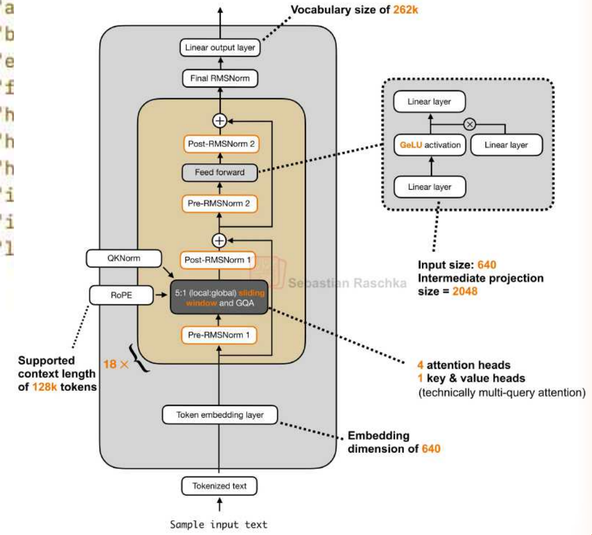


main / gemma-3-270m / config.json

sindhuraghuram uploading the PT weights for Gemma 3 2

raw | Copy download link | history | blame | contribute

```
{
  "_sliding_window_pattern": 6,
  "architectures": [
    "Gemma3ForCausalLM"
  ],
  "attention_bias": false,
  "a
  "a
  "b
  "e
  "f
  "h
  "h
  "h
  "i
  "i
  "l
```



6 Code up the architecture (starting from a similar architecture)

```
{
  "architectures": [
    "Qwen3ForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "bos_token_id": 151643,
  "eos_token_id": 151645,
  "head_dim": 128,
  "hidden_act": "silu",
  "hidden_size": 4096,
```

Related architecture (as template)

main / LLMs-from-scratch / ch05 / 11_qwen3 / standalone-qwen3.ipynb

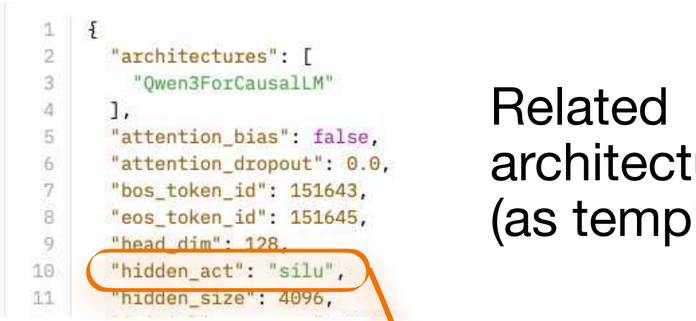
Preview | Code | Blame | 1238 lines (1238 loc) · 45.1 KB

1. Architecture code

In [24]:

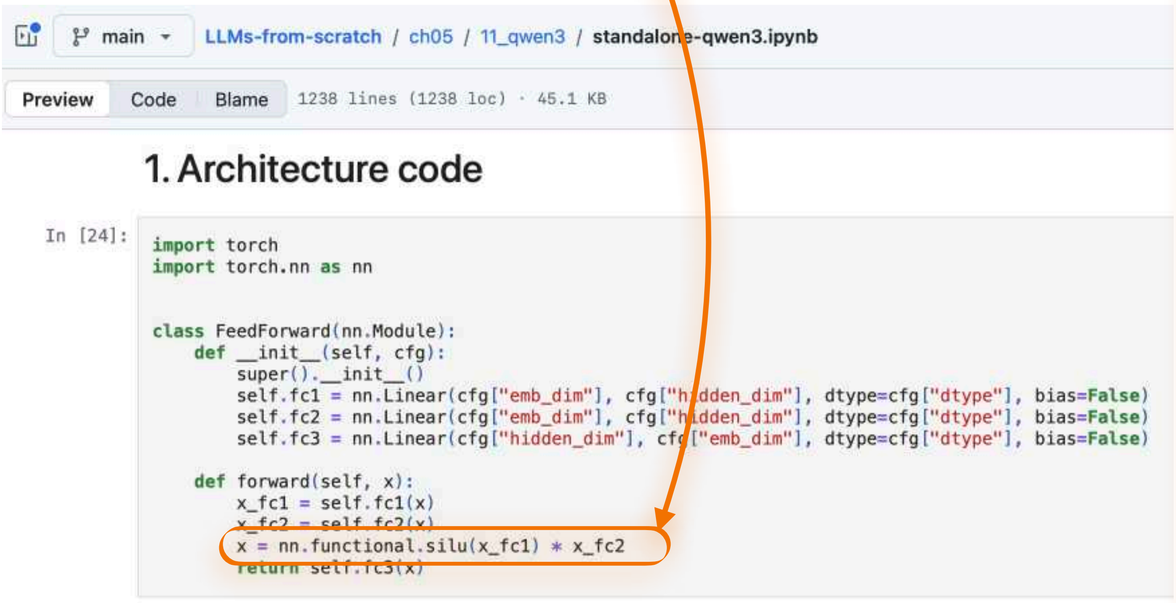
```
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.silu(x_fc1) * x_fc2
        return self.fc3(x)
```

6 Code up the architecture (starting from a similar architecture)



```
{
  "architectures": [
    "Qwen3ForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "bos_token_id": 151643,
  "eos_token_id": 151645,
  "head_dim": 128,
  "hidden_act": "silu",
  "hidden_size": 4096,
```

Related architecture (as template)



main LLMs-from-scratch / ch05 / 11_qwen3 / standalone-qwen3.ipynb

Preview Code Blame 1238 lines (1238 loc) · 45.1 KB

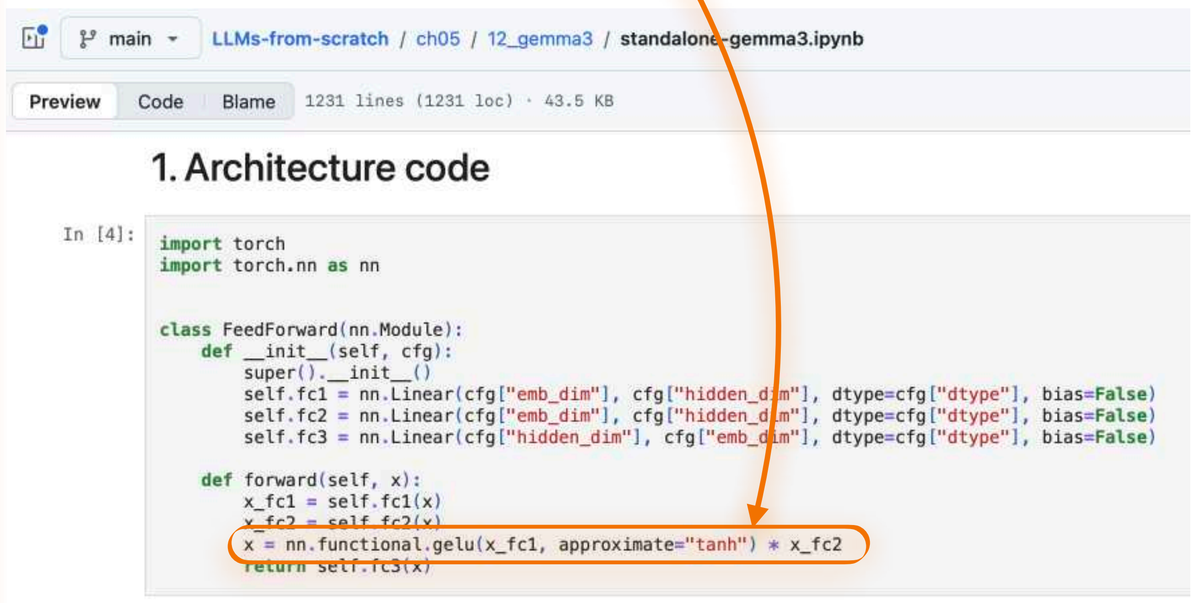
1. Architecture code

In [24]:

```
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.silu(x_fc1) * x_fc2
        return self.fc3(x)
```

```
{
  "_sliding_window_pattern": 6,
  "architectures": [
    "Gemma3ForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "attn_logit_softcapping": null,
  "bos_token_id": 2,
  "eos_token_id": 1,
  "final_logit_softcapping": null,
  "head_dim": 256,
  "hidden_activation": "gelu_pytorch_tanh",
  "hidden_size": 640,
```

New architecture to implement



main LLMs-from-scratch / ch05 / 12_gemma3 / standalone-gemma3.ipynb

Preview Code Blame 1231 lines (1231 loc) · 43.5 KB

1. Architecture code

In [4]:

```
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.gelu(x_fc1, approximate="tanh") * x_fc2
        return self.fc3(x)
```

7 Check if it generates coherent text

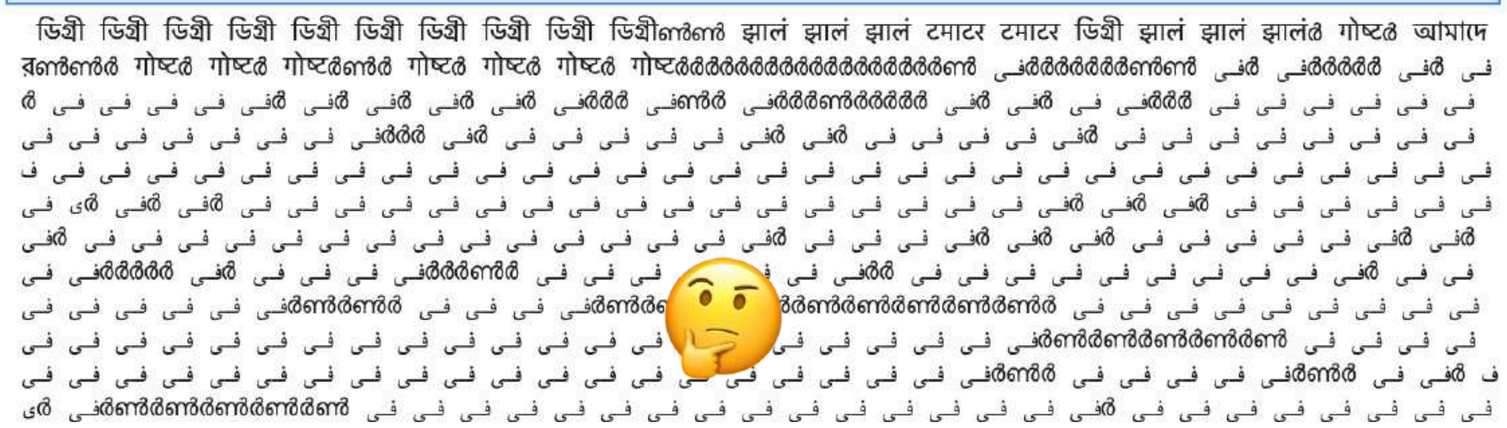
```
prompt = "Give me a short introduction to large language models.
```

```
for token in generate_text_basic_stream(
    model=model,
    token_ids=input_token_ids_tensor,
    max_new_tokens=500,
    eos_token_id=tokenizer.end_of_turn_token_id
):
    token_id = token.squeeze(0).tolist()
    print(
        tokenizer.decode(token_id),
        end="",
        flush=True
    )
```

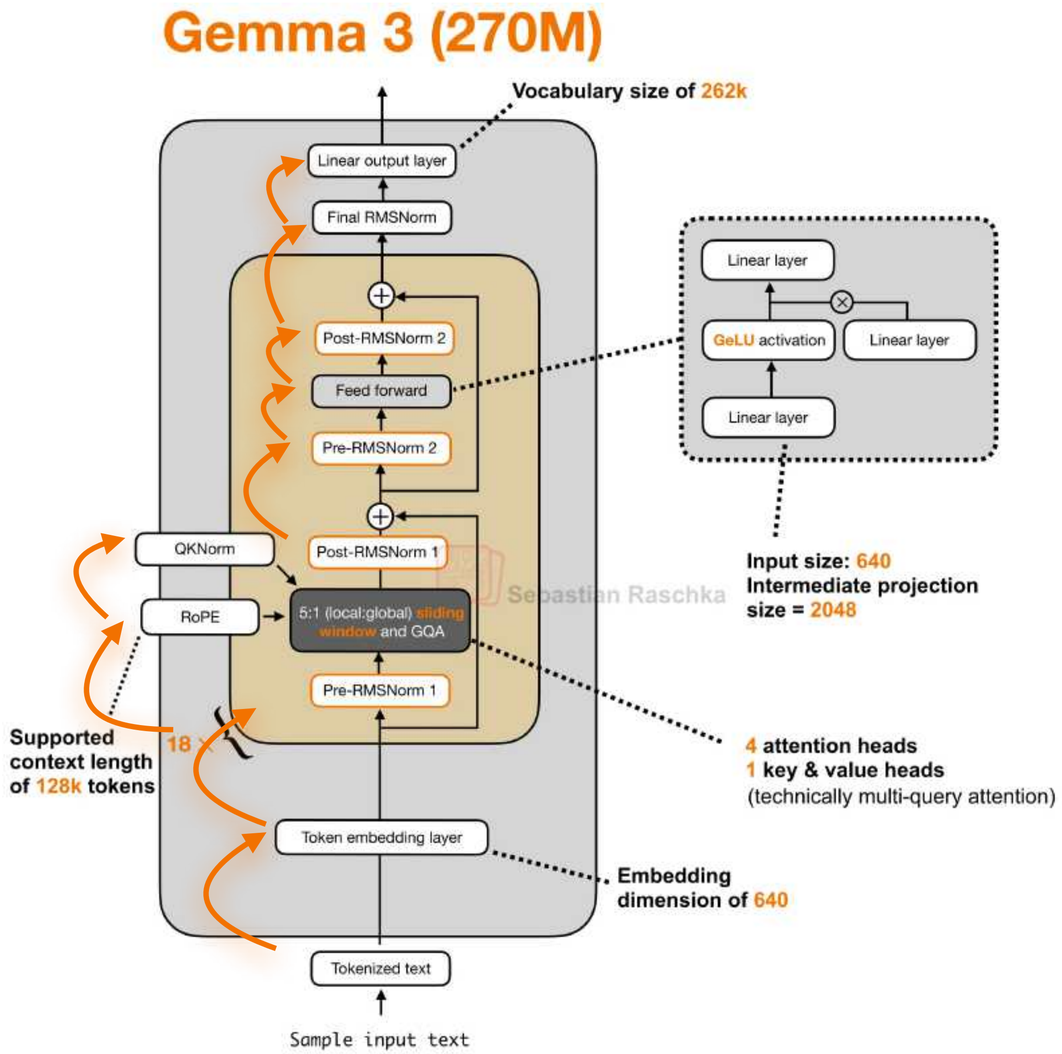
7 Check if it generates coherent text

```
prompt = "Give me a short introduction to large language models.
```

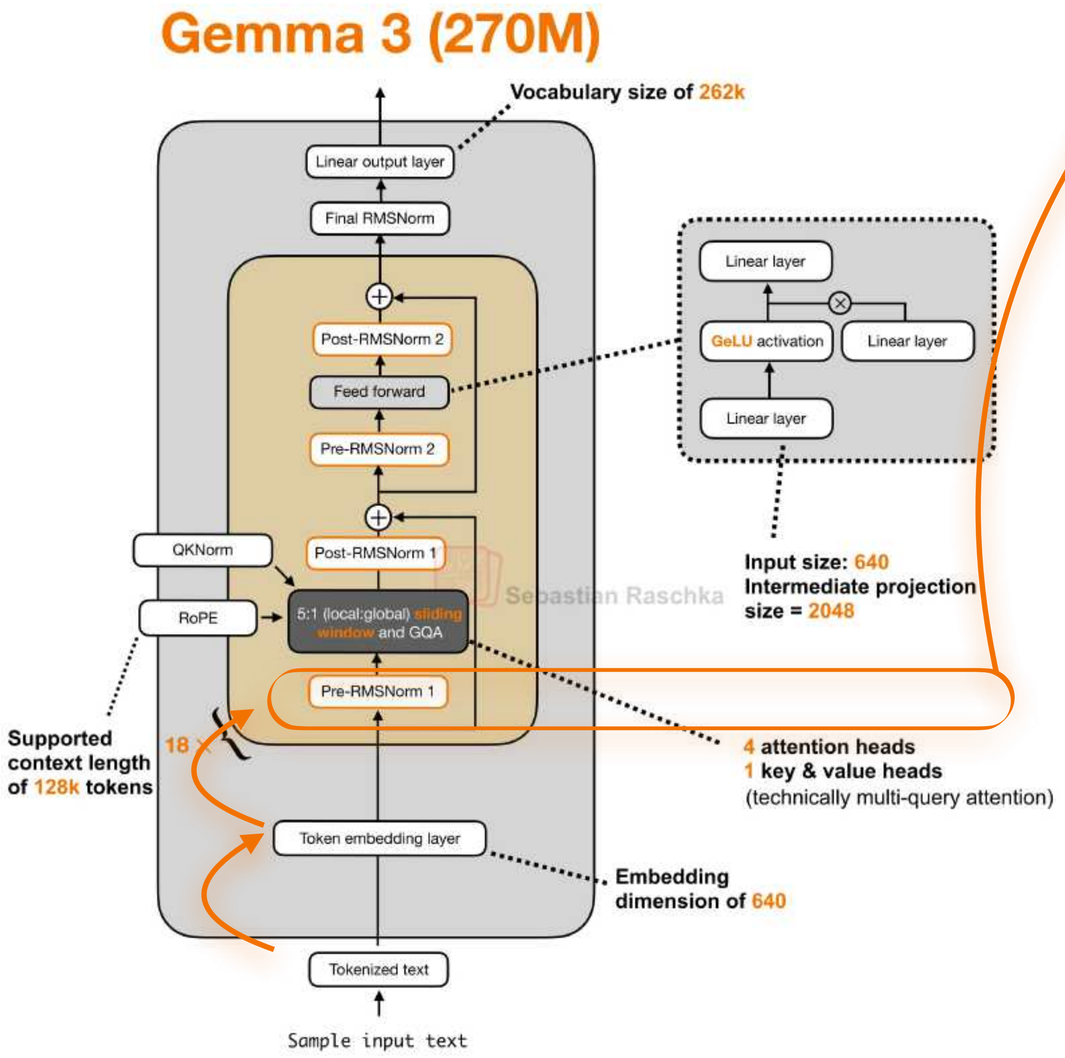
```
for token in generate_text_basic_stream(
    model=model,
    token_ids=input_token_ids_tensor,
    max_new_tokens=500,
    eos_token_id=tokenizer.end_of_turn_token_id
):
    token_id = token.squeeze(0).tolist()
    print(
        tokenizer.decode(token_id),
        end="",
        flush=True
    )
```



8 Check layer by layer to find the mistake



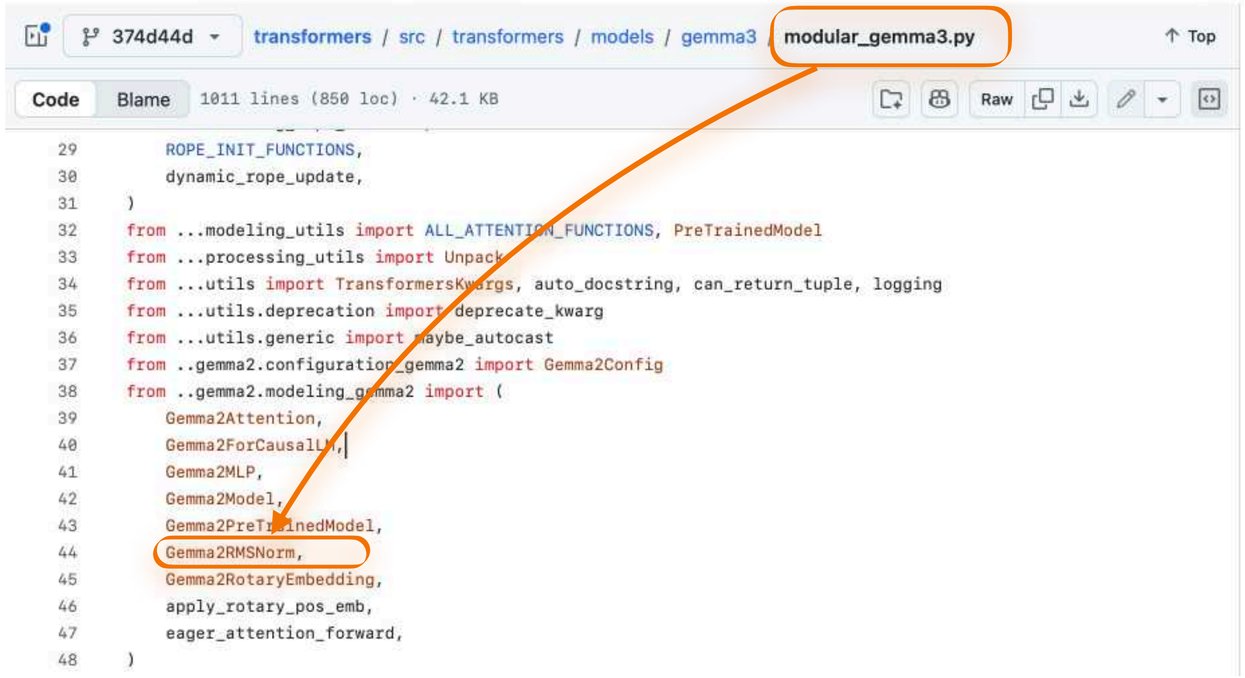
8 Check layer by layer to find the mistake




```
12_gemma3 — sebastian@Mac — ..h05/12_gemma3 — -zsh — 107×37
→  12_gemma3 git:(main) ✗ uv run tests/gemma3_layer_debugger-v1.py
[OK] embedding: max=0.00e+00, mean=0.00e+00
[DIFF] block_0: max=2.98e+00, mean=8.80e-01
  [DIFF] block_0.input_layernorm: max=2.34e+00, mean=8.11e-01
  [DIFF] block_0.att.q_proj: max=3.45e-01, mean=9.00e-02
  [DIFF] block_0.att.k_proj: max=2.99e-01, mean=9.04e-02
  [DIFF] block_0.att.v_proj: max=3.84e-01, mean=8.81e-02
  [DIFF] block_0.att.q_norm: max=2.42e+00, mean=8.18e-01
  [DIFF] block_0.att.k_norm: max=2.17e+00, mean=8.26e-01
  [DIFF] block_0.att.o_proj: max=5.82e-02, mean=5.32e-03
  [DIFF] block_0.att: max=5.82e-02, mean=5.32e-03
  [DIFF] block_0.post_attention_layernorm: max=3.02e+00, mean=7.91e-01
  [DIFF] block_0.pre_feedforward_layernorm: max=2.97e+00, mean=8.18e-01
  [DIFF] block_0.ff.gate_proj: max=3.32e-01, mean=8.97e-02
  [DIFF] block_0.ff.up_proj: max=3.29e-01, mean=8.94e-02
  [DIFF] block_0.ff.down_proj: max=2.14e-03, mean=6.63e-04
  [DIFF] block_0.ff: max=2.14e-03, mean=6.63e-04
  [DIFF] block_0.post_feedforward_layernorm: max=1.54e+00, mean=5.08e-01
[DIFF] block_1: max=4.13e+00, mean=1.29e+00
  [DIFF] block_1.input_layernorm: max=2.79e+00, mean=8.04e-01
  [DIFF] block_1.att.q_proj: max=3.82e-01, mean=1.05e-01
  [DIFF] block_1.att.k_proj: max=3.35e-01, mean=1.05e-01
  [DIFF] block_1.att.v_proj: max=3.37e-01, mean=9.21e-02
  [DIFF] block_1.att.q_norm: max=2.21e+00, mean=8.13e-01
  [DIFF] block_1.att.k_norm: max=2.29e+00, mean=8.40e-01
  [DIFF] block_1.att.o_proj: max=3.15e-02, mean=7.69e-03
  [DIFF] block_1.att: max=3.15e-02, mean=7.69e-03
  [DIFF] block_1.post_attention_layernorm: max=2.60e+00, mean=7.89e-01
  [DIFF] block_1.pre_feedforward_layernorm: max=3.02e+00, mean=7.75e-01
  [DIFF] block_1.ff.gate_proj: max=5.03e-01, mean=9.30e-02
  [DIFF] block_1.ff.up_proj: max=3.09e-01, mean=9.14e-02
  [DIFF] block_1.ff.down_proj: max=2.95e-03, mean=8.99e-04
  [DIFF] block_1.ff: max=2.95e-03, mean=8.99e-04
  [DIFF] block_1.post_feedforward_layernorm: max=1.80e+00, mean=6.04e-01
[DIFF] final_norm: max=2.52e+00, mean=8.13e-01
[DIFF] logits: max=4.05e-01, mean=9.47e-02
→  12_gemma3 git:(main) ✗
```

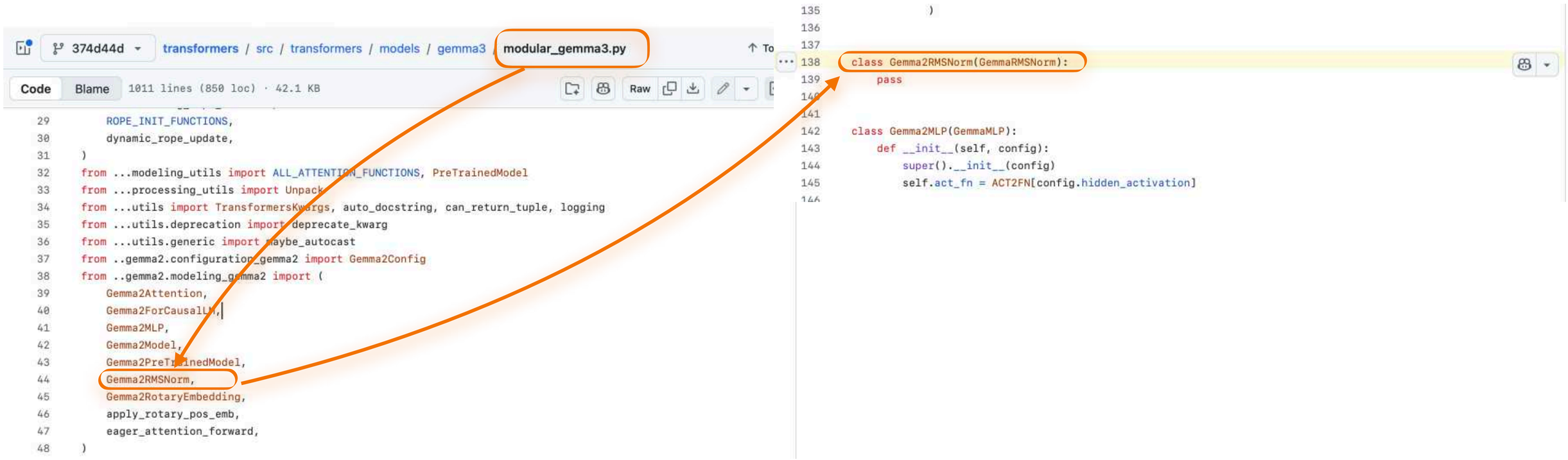
*Against HF transformers reference implementation; small config with random weights

9 Read reference implementation code



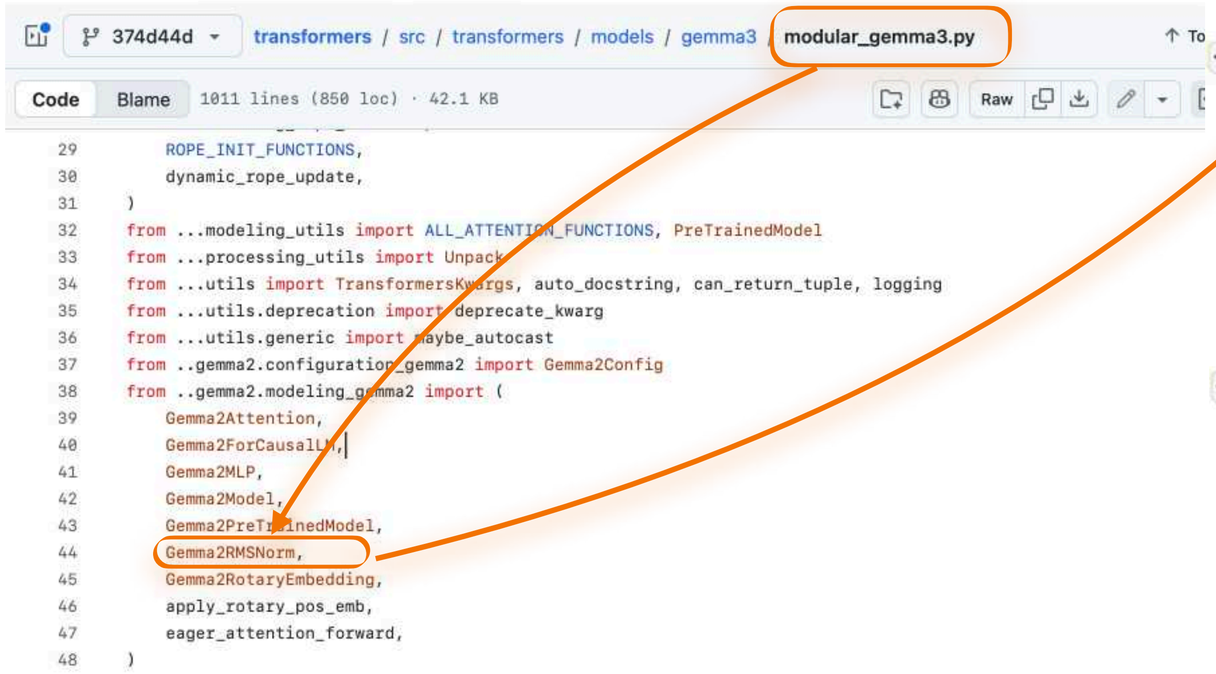
…sometimes a bit like an Easter egg hunt

9 Read reference implementation code



…sometimes a bit like an Easter egg hunt

9 Read reference implementation code



…sometimes a bit like an Easter egg hunt

10 Implement the architecture specifics

main LLMs-from-scratch / ch05 / 11_qwen3 / standalone-qwen3.ipynb

Preview Code Blame 1238 lines (1238 loc) · 45.1 KB

1. Architecture code

In [24]:

```python
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.silu(x_fc1) * x_fc2
        return self.fc3(x)
```

In [25]:

```python
class RMSNorm(nn.Module):
    def __init__(self, emb_dim, eps=1e-6, bias=False, qwen3_compatible=True):
        super().__init__()
        self.eps = eps
        self.qwen3_compatible = qwen3_compatible
        self.scale = nn.Parameter(torch.ones(emb_dim))
        self.shift = nn.Parameter(torch.zeros(emb_dim)) if bias else None

    def forward(self, x):
        input_dtype = x.dtype

        if self.qwen3_compatible:
            x = x.to(torch.float32)

        variance = x.pow(2).mean(dim=-1, keepdim=True)
        norm_x = x * torch.rsqrt(variance + self.eps)
        norm_x = norm_x * self.scale

        if self.shift is not None:
            norm_x = norm_x + self.shift

        return norm_x.to(input_dtype)
```

adjust & “learn”

main LLMs-from-scratch / ch05 / 12_gemma3 / standalone-gemma3.ipynb

Preview Code Blame 1231 lines (1231 loc) · 43.5 KB

1. Architecture code

In [4]:

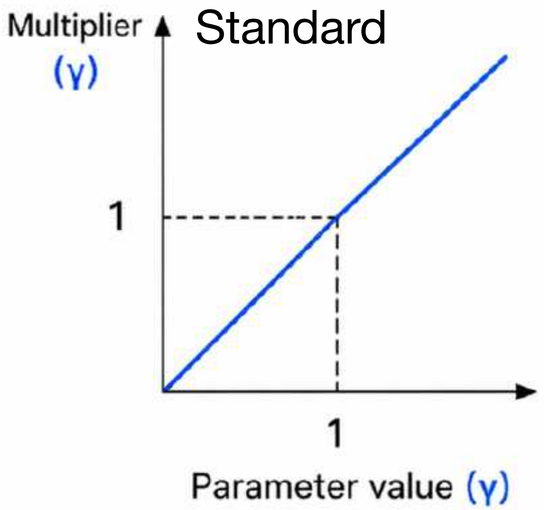
```python
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.gelu(x_fc1, approximate="tanh") * x_fc2
        return self.fc3(x)
```

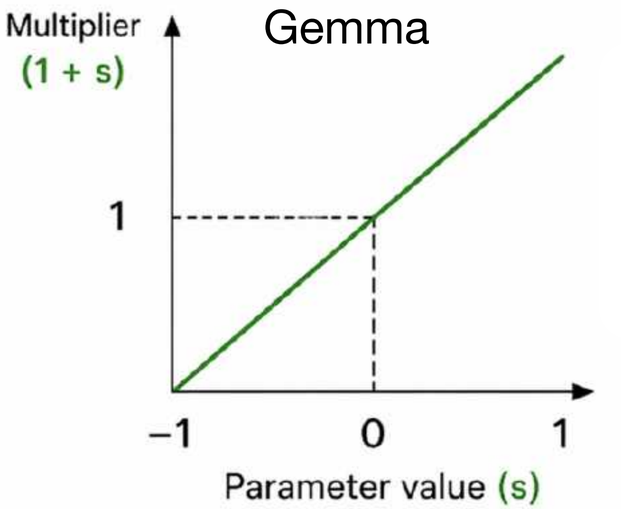
In [5]:

```python
class RMSNorm(nn.Module):
    def __init__(self, emb_dim, eps=1e-6, bias=False):
        super().__init__()
        self.eps = eps
        # Gemma3 stores zero-centered weights and uses (1 + weight) during forward
        self.scale = nn.Parameter(torch.zeros(emb_dim))
        self.shift = nn.Parameter(torch.zeros(emb_dim)) if bias else None

    def forward(self, x):
        # Match HF Gemma3: compute norm in float32, then scale by (1 + w)
        input_dtype = x.dtype
        x_f = x.float()
        var = x_f.pow(2).mean(dim=-1, keepdim=True)
        x_norm = x_f * torch.rsqrt(var + self.eps)
        out = x_norm * (1.0 + self.scale.float())

        if self.shift is not None:
            out = out + self.shift.float()

        return out.to(input_dtype)
```

Side note:

- standard RMSNorm learns the scale directly
- Gemma RMSNorm learns an offset from 1 (zero-centered, more stable learning)

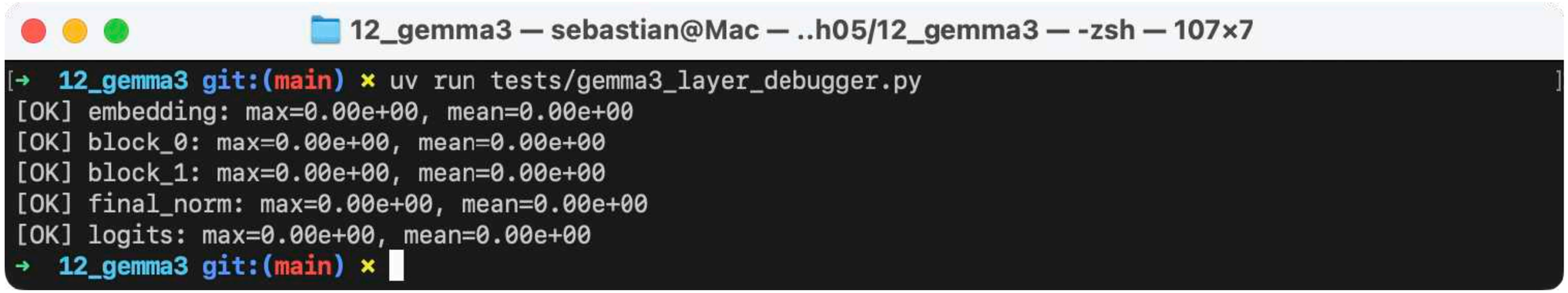





```
y = (x / rms(x)) * scale
```

```
y = (x / rms(x)) * (1 + scale)
```

11 Run layer check until everything matches



*Against HF transformers reference implementation;
small config with random weights

12 Check if model generates same text as reference

1. Architecture code

```python
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.gelu(x_fc1, approximate="tanh") * x_fc2
        return self.fc3(x)
```

```python
class RMSNorm(nn.Module):
    def __init__(self, emb_dim, eps=1e-6, bias=False):
        super().__init__()
        self.eps = eps
        # Gemma3 stores zero-centered weights and uses (1 + weight) during forward
        self.scale = nn.Parameter(torch.zeros(emb_dim))
        self.shift = nn.Parameter(torch.zeros(emb_dim)) if bias else None

    def forward(self, x):
```

```python
prompt = "Give me a short introduction to large language models.
```

```python
for token in generate_text_basic_stream(
    model=model,
    token_ids=input_token_ids_tensor,
    max_new_tokens=500,
    eos_token_id=tokenizer.end_of_turn_token_id
):
    token_id = token.squeeze(0).tolist()
    print(
        tokenizer.decode(token_id),
        end="",
        flush=True
    )
```

```
Large language models (LLMs) are sophisticated artificial intelligence systems that can understand,
generate, and manipulate human language. They are trained on massive amounts of text data to learn p
atterns and relationships within language, enabling them to perform a wide range of tasks, from writ
ing articles and answering questions to translating languages and summarizing information.
```

12 Check if model generates same text as reference

LLMs-from-scratch / ch05 / 12_gemma3 / standalone-gemma3.ipynb

Code Blame Raw

1. Architecture code

```python
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.gelu(x_fc1, approximate="tanh") * x_fc2
        return self.fc3(x)
```

```python
class RMSNorm(nn.Module):
    def __init__(self, emb_dim, eps=1e-6, bias=False):
        super().__init__()
        self.eps = eps
        # Gemma3 stores zero-centered weights and uses (1 + weight) during forward
        self.scale = nn.Parameter(torch.zeros(emb_dim))
        self.shift = nn.Parameter(torch.zeros(emb_dim)) if bias else None

    def forward(self, x):
```

```python
prompt = "Give me a short introduction to large language models."
```

```python
for token in generate_text_basic_stream(
    model=model,
    token_ids=input_token_ids_tensor,
    max_new_tokens=500,
    eos_token_id=tokenizer.end_of_turn_token_id
):
    token_id = token.squeeze(0).tolist()
    print(
        tokenizer.decode(token_id),
        end="",
        flush=True
    )
```

```
Large language models (LLMs) are sophisticated artificial intelligence systems that can understand, generate, and manipulate human language. They are trained on massive amounts of text data to learn p atterns and relationships within language, enabling them to perform a wide range of tasks, from writ ing articles and answering questions to translating languages and summarizing information.
```

*Against HF transformers reference implementation; now the full architecture with pre-trained weights

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model_id = "google/gemma-3-270m-it"
prompt = "Give me a short introduction to large language models."

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
model.generation_config.do_sample = False
model.generation_config.top_p = None
model.generation_config.top_k = None
model.generation_config.pad_token_id = tokenizer.pad_token_id
model.eval();
```

```
Loading weights: 100% ████████████ 236/236 [00:00<00:00, 3884.39it/s, Materializing param=model.norm.weight]
```

```python
messages = [{"role": "user", "content": prompt}]

inputs = tokenizer.apply_chat_template(
    messages,
    tokenize=True,
    add_generation_prompt=True,
    return_tensors="pt",
)

outputs = model.generate(
    **inputs,
    max_new_tokens=500,
    do_sample=False,
    num_beams=1,
    pad_token_id=tokenizer.pad_token_id,
)

response = tokenizer.decode(
    outputs[0][inputs["input_ids"].shape[-1]:],
    skip_special_tokens=True,
)
print(response)
```

```
Large language models (LLMs) are sophisticated artificial intelligence systems that can understand, generate, and manipulate human language. They are trained on massive amounts of text data to learn p atterns and relationships within language, enabling them to perform a wide range of tasks, from writ ing articles and answering questions to translating languages and summarizing information.
```

12 Check if model generates same text as reference

LLMs-from-scratch / ch05 / 12_gemma3 / standalone-gemma3.ipynb

Code Blame Raw

1. Architecture code

```python
import torch
import torch.nn as nn


class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc2 = nn.Linear(cfg["emb_dim"], cfg["hidden_dim"], dtype=cfg["dtype"], bias=False)
        self.fc3 = nn.Linear(cfg["hidden_dim"], cfg["emb_dim"], dtype=cfg["dtype"], bias=False)

    def forward(self, x):
        x_fc1 = self.fc1(x)
        x_fc2 = self.fc2(x)
        x = nn.functional.gelu(x_fc1, approximate="tanh") * x_fc2
        return self.fc3(x)
```

```python
class RMSNorm(nn.Module):
    def __init__(self, emb_dim, eps=1e-6, bias=False):
        super().__init__()
        self.eps = eps
        # Gemma3 stores zero-centered weights and uses (1 + weight) during forward
        self.scale = nn.Parameter(torch.zeros(emb_dim))
        self.shift = nn.Parameter(torch.zeros(emb_dim)) if bias else None

    def forward(self, x):
```

```python
prompt = "Give me a short introduction to large language models."
```

```python
for token in generate_text_basic_stream(
    model=model,
    token_ids=input_token_ids_tensor,
    max_new_tokens=500,
    eos_token_id=tokenizer.end_of_turn_token_id
):
    token_id = token.squeeze(0).tolist()
    print(
        tokenizer.decode(token_id),
        end="",
        flush=True
    )
```

```
Large language models (LLMs) are sophisticated artificial intelligence systems that can understand, generate, and manipulate human language. They are trained on massive amounts of text data to learn p atterns and relationships within language, enabling them to perform a wide range of tasks, from writ ing articles and answering questions to translating languages and summarizing information.
```

✅

*Against HF transformers reference implementation; now the full architecture with pre-trained weights

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model_id = "google/gemma-3-270m-it"
prompt = "Give me a short introduction to large language models."

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
model.generation_config.do_sample = False
model.generation_config.top_p = None
model.generation_config.top_k = None
model.generation_config.pad_token_id = tokenizer.pad_token_id
model.eval();
```

```
Loading weights: 100% ████████████ 236/236 [00:00<00:00, 3884.39it/s, Materializing param=model.norm.weight]
```

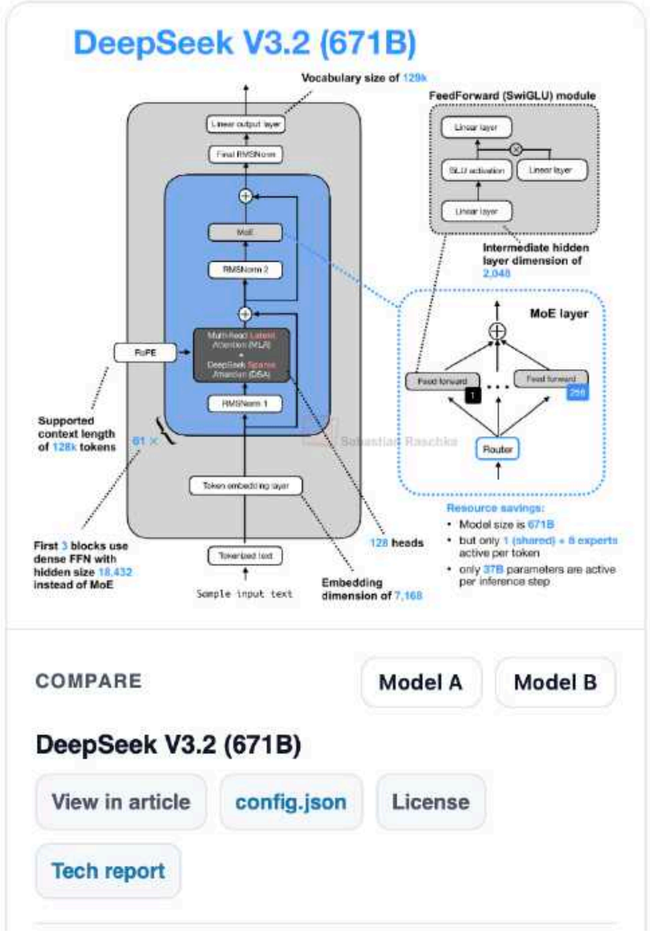
```python
messages = [{"role": "user", "content": prompt}]

inputs = tokenizer.apply_chat_template(
    messages,
    tokenize=True,
    add_generation_prompt=True,
    return_tensors="pt",
)

outputs = model.generate(
    **inputs,
    max_new_tokens=500,
    do_sample=False,
    num_beams=1,
    pad_token_id=tokenizer.pad_token_id,
)

response = tokenizer.decode(
    outputs[0][inputs["input_ids"].shape[-1]:],
    skip_special_tokens=True,
)
print(response)
```

```
Large language models (LLMs) are sophisticated artificial intelligence systems that can understand, generate, and manipulate human language. They are trained on massive amounts of text data to learn p atterns and relationships within language, enabling them to perform a wide range of tasks, from writ ing articles and answering questions to translating languages and summarizing information.
```

From Code to Gallery

https://llm-gallery.com/

Architecture summaries



COMPARE

Model A | Model B

DeepSeek V3.2 (671B)

View in article | config.json | License

Tech report

DeepSeek's successor keeps the V3 template but adds sparse attention to cut long-context costs.

SCALE	CONTEXT (TOKENS)
671B total, 37B active (5.5% active)	128,000
LICENSE	**DATE**
MIT License	2025-12-01
DECODER TYPE	**ATTENTION**
Sparse MoE	MLA with DeepSeek Sparse Attention
LAYER MIX	**KV CACHE / TOKEN (BF16)** info
61 MLA	68.6 KiB · Low

KEY DETAIL

An evolutionary update focused on efficiency rather than a new base layout.

AA INTELLIGENCE INDEX info

Total score 32.1

GENERAL	SCIENTIFIC	CODING	AGENTS
29.7	24.2	34.6	39.8

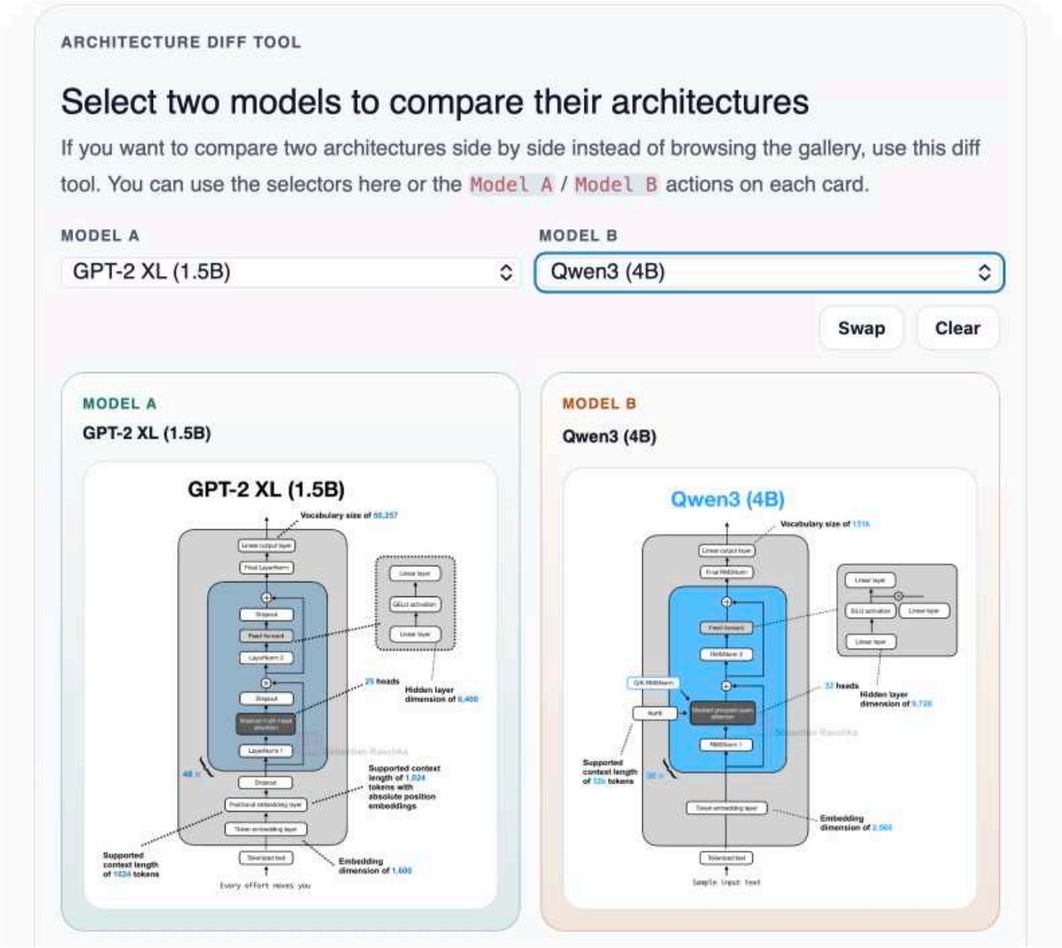
Show less

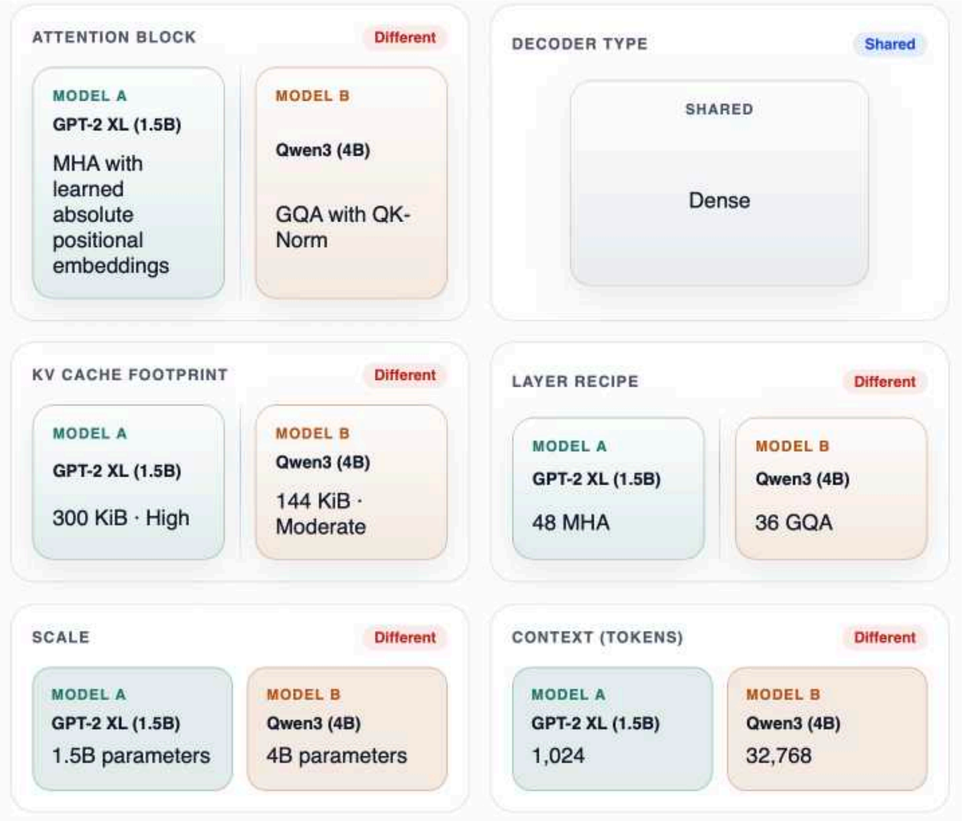
RELATED CONCEPTS

MLA | MoE | DeepSeek Sparse Attention

https://llm-gallery.com/

Architecture comparisons





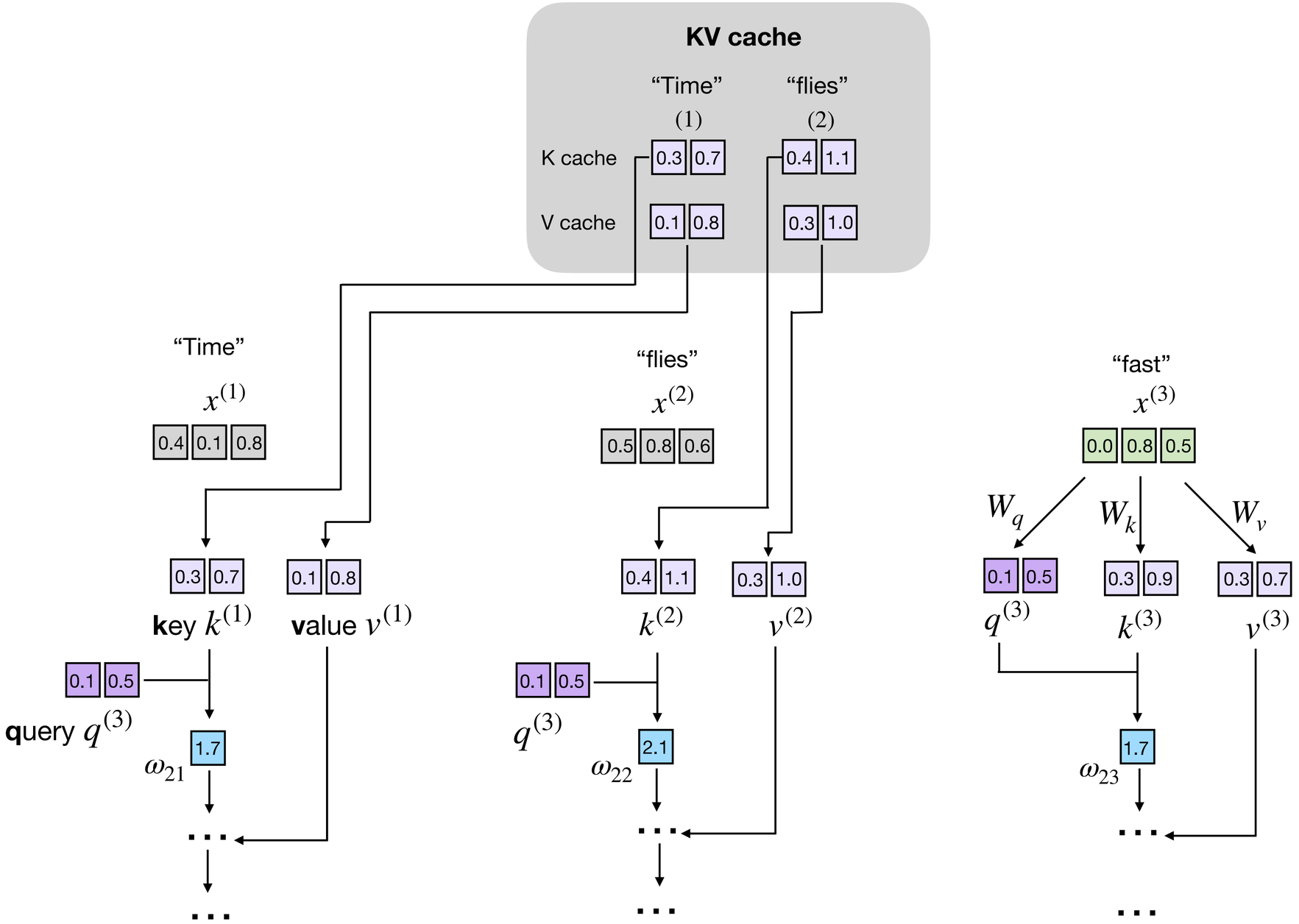
https://llm-gallery.com/

Lessons and trends from 50+ models

Main focus: strategies for reducing KV cache size

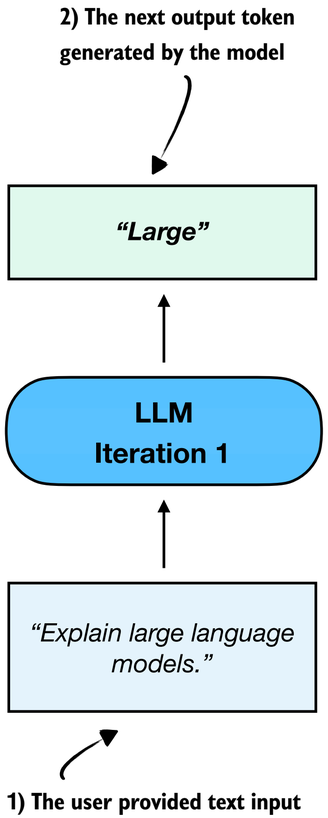
Main focus: strategies for reducing KV cache size

Because we need longer contexts (reasoning, agent harnesses)

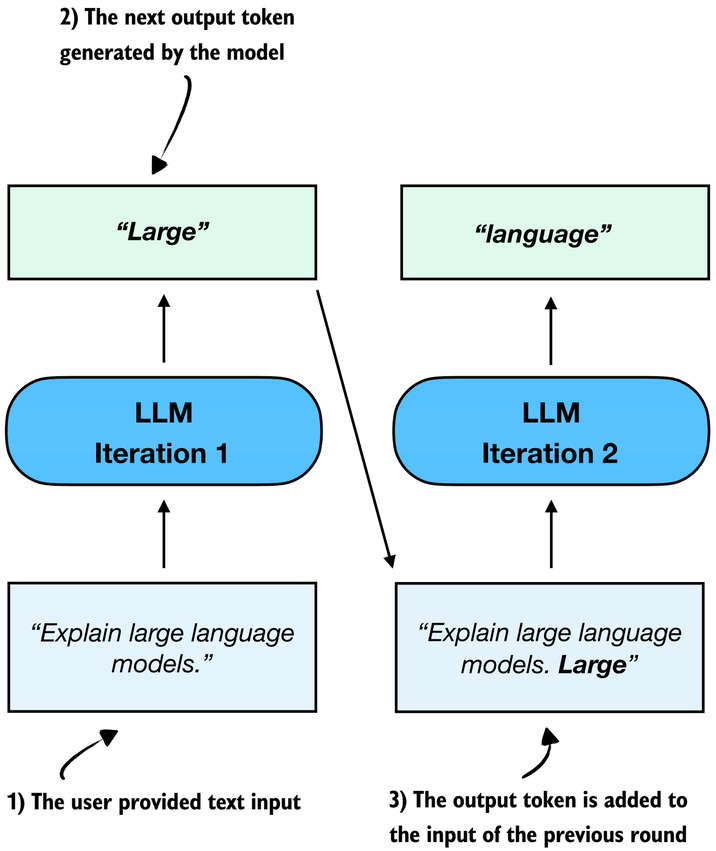


https://magazine.sebastianraschka.com/p/coding-the-kv-cache-in-llms

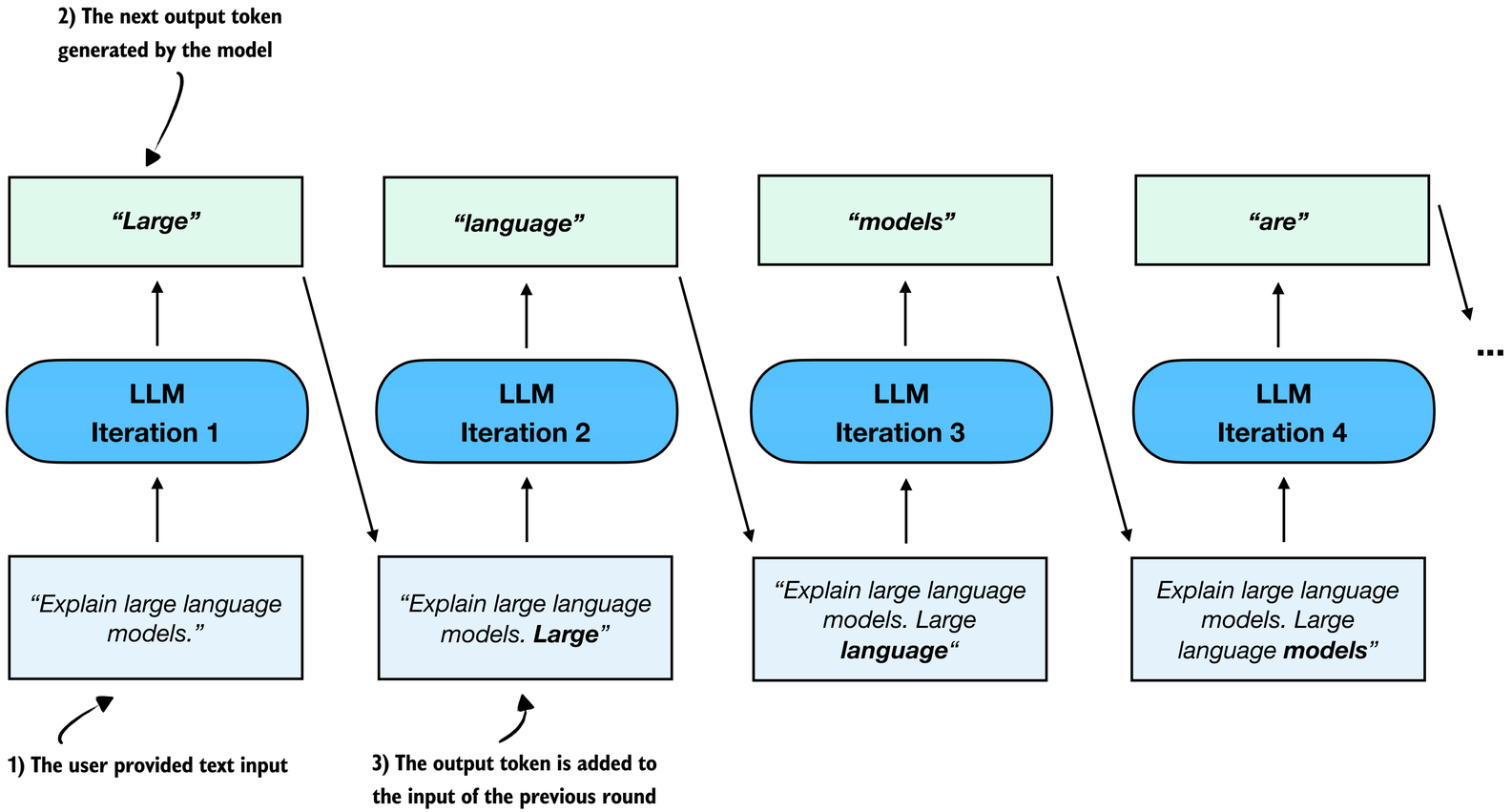
Text generation without KV cache



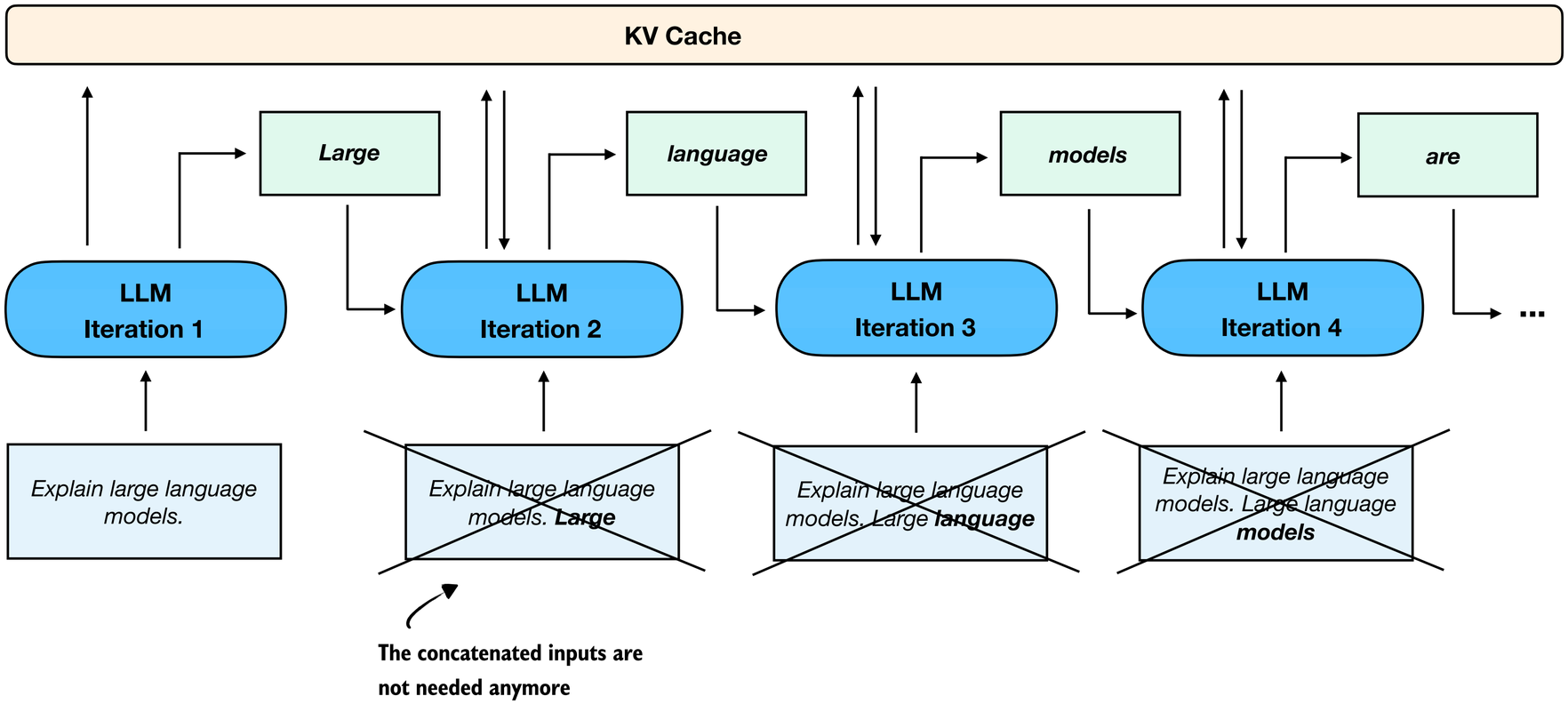
Text generation without KV cache



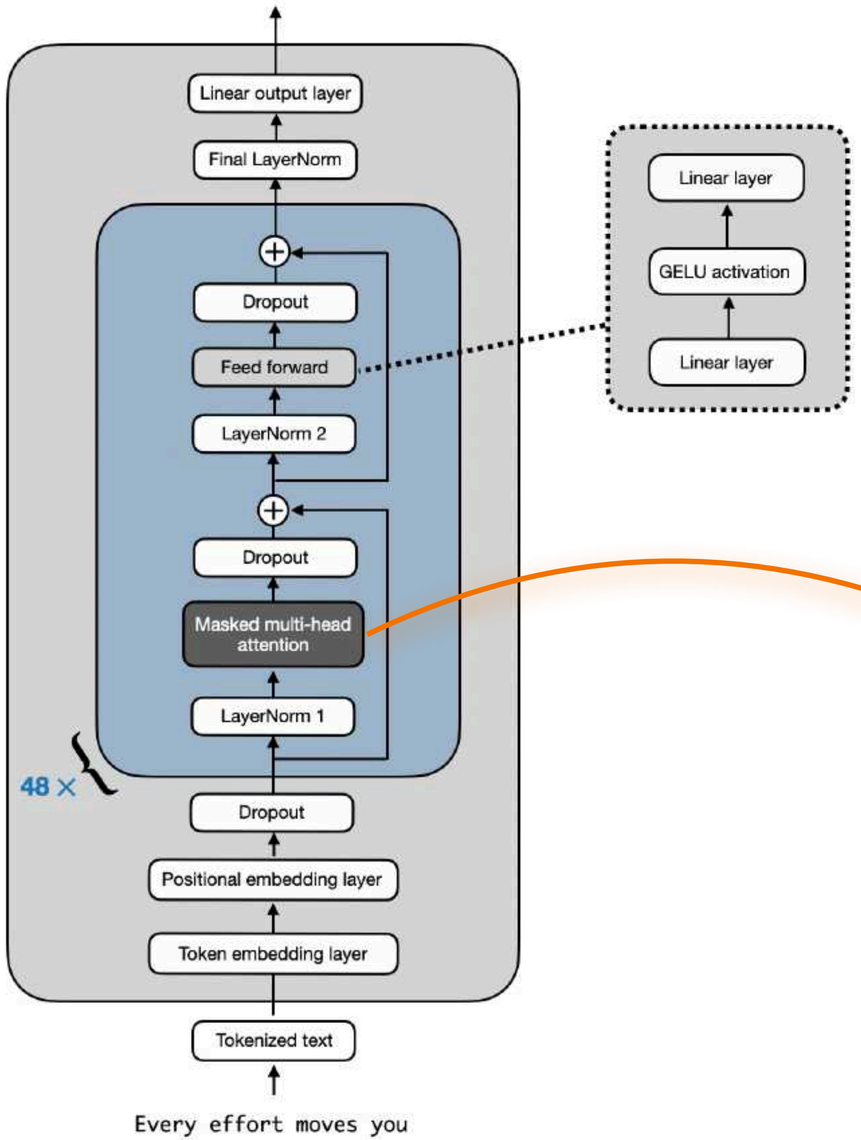
Text generation without KV cache



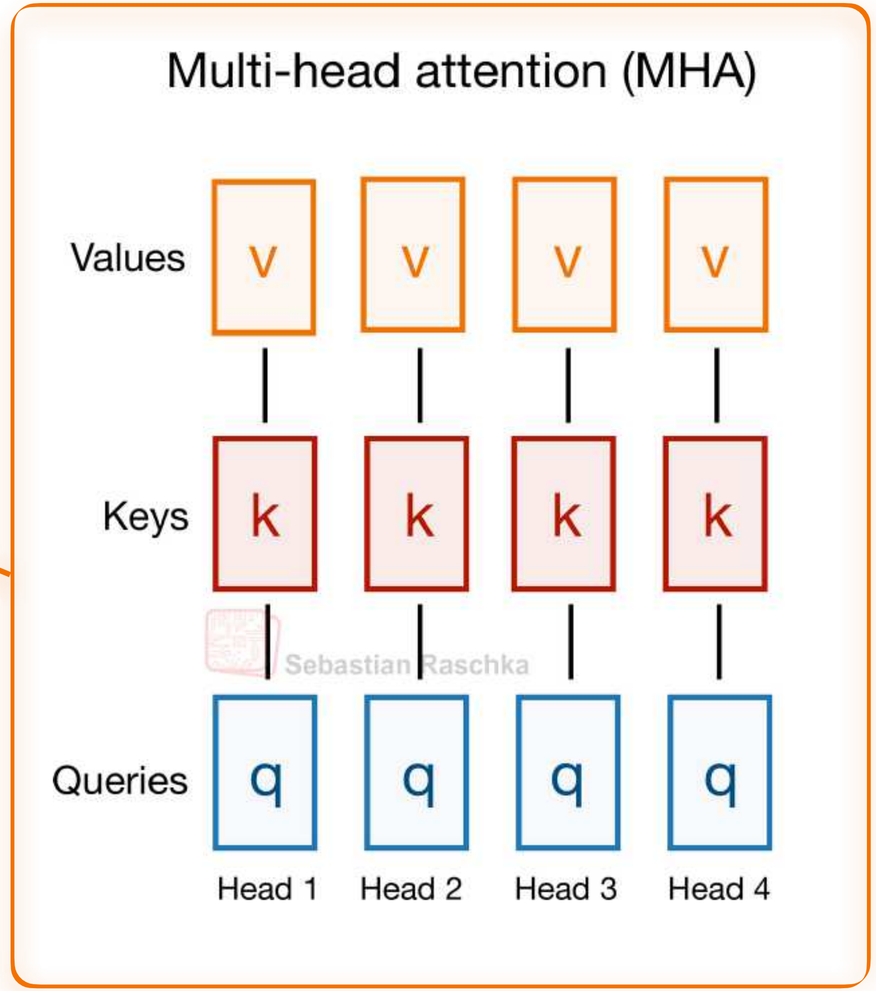
Text generation with KV cache



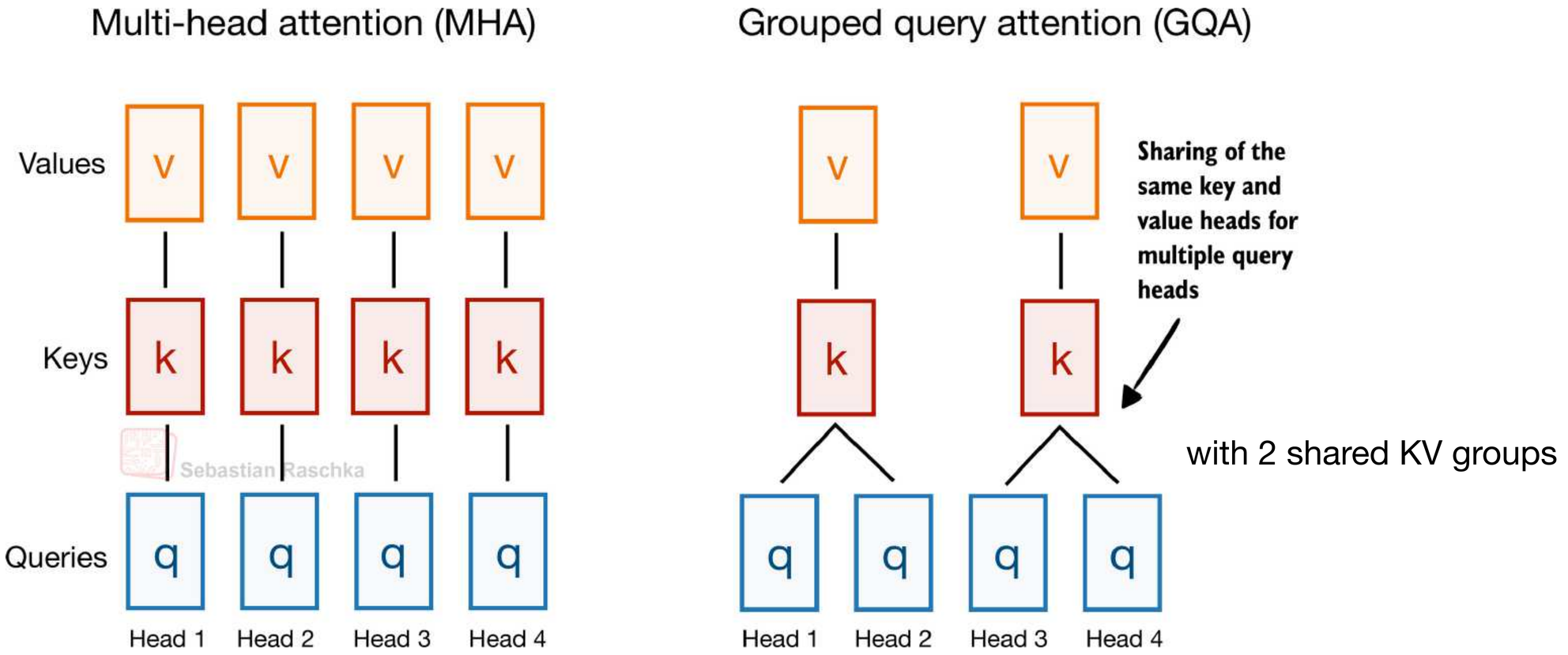
GPT-2 XL (1.5B)


Linear output layer
Final LayerNorm
Dropout
Feed forward
LayerNorm 2
Dropout
Masked multi-head attention
LayerNorm 1
48 ×
Dropout
Positional embedding layer
Token embedding layer
Tokenized text
Every effort moves you

Linear layer
GELU activation
Linear layer

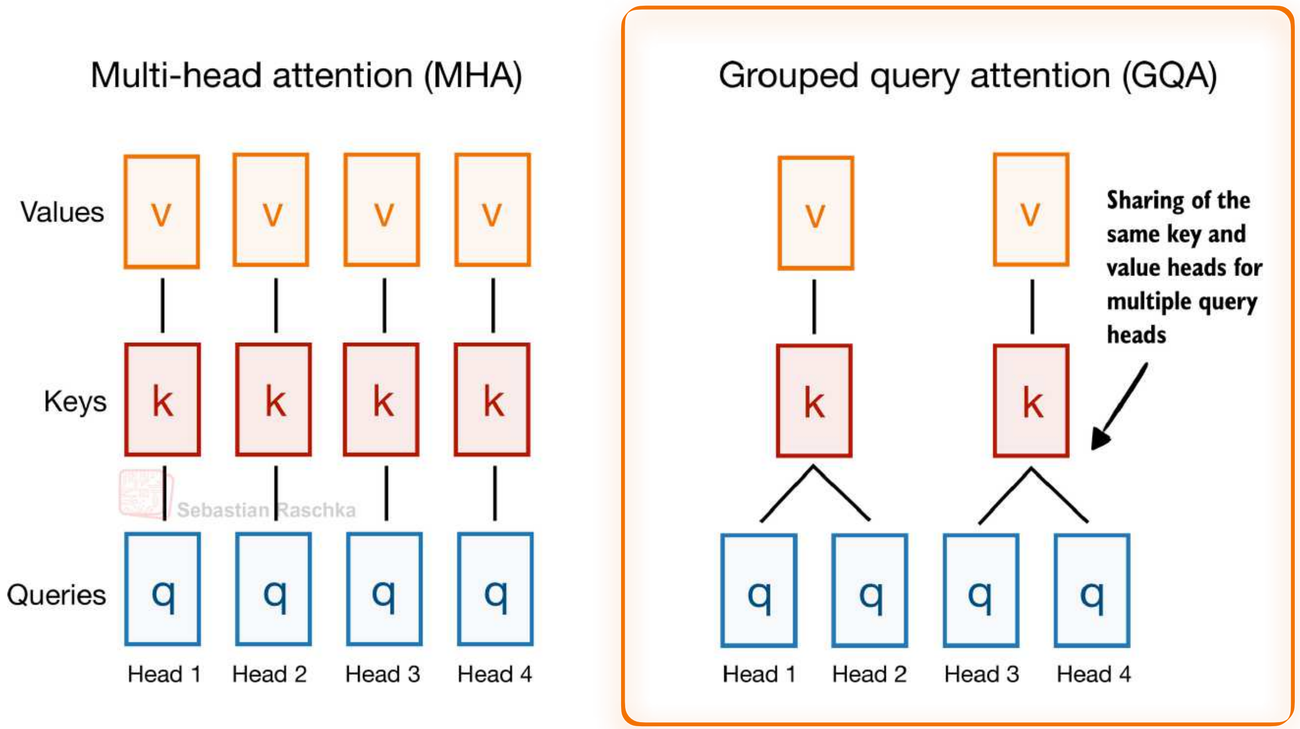

Multi-head attention (MHA)
Values
Keys
Queries
v
k
q
Head 1
Head 2
Head 3
Head 4
Sebastian Raschka

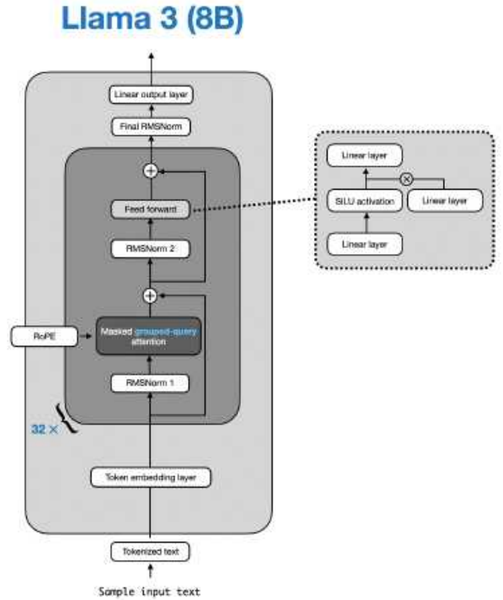
Method 1 to reduce KV cache size

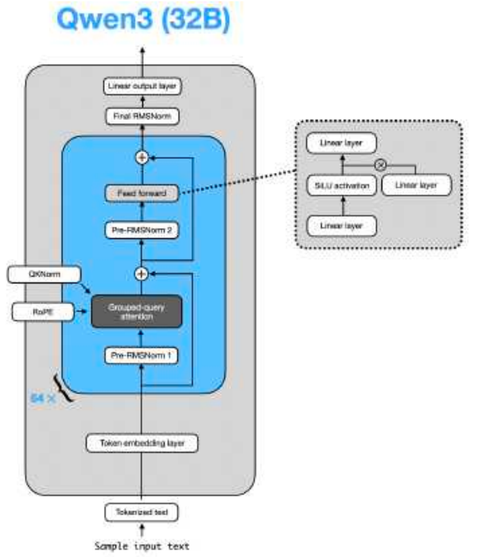


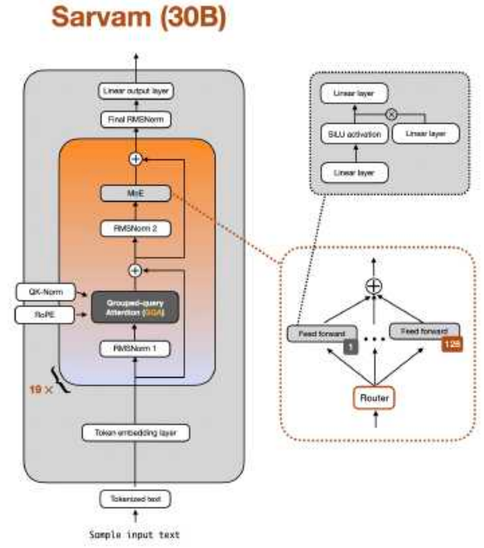
https://magazine.sebastianraschka.com/p/visual-attention-variants

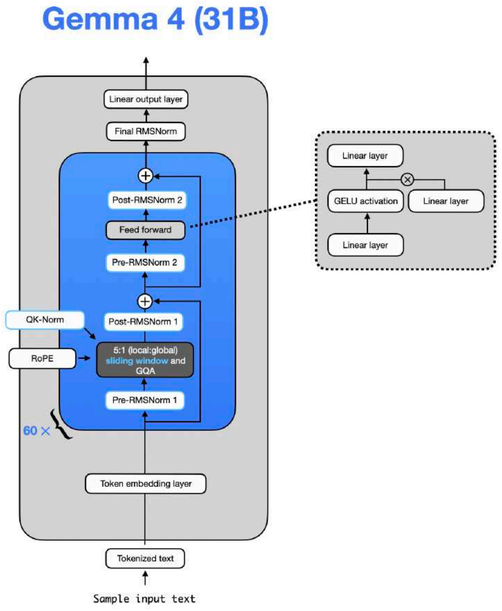
Method 1 to reduce KV cache size







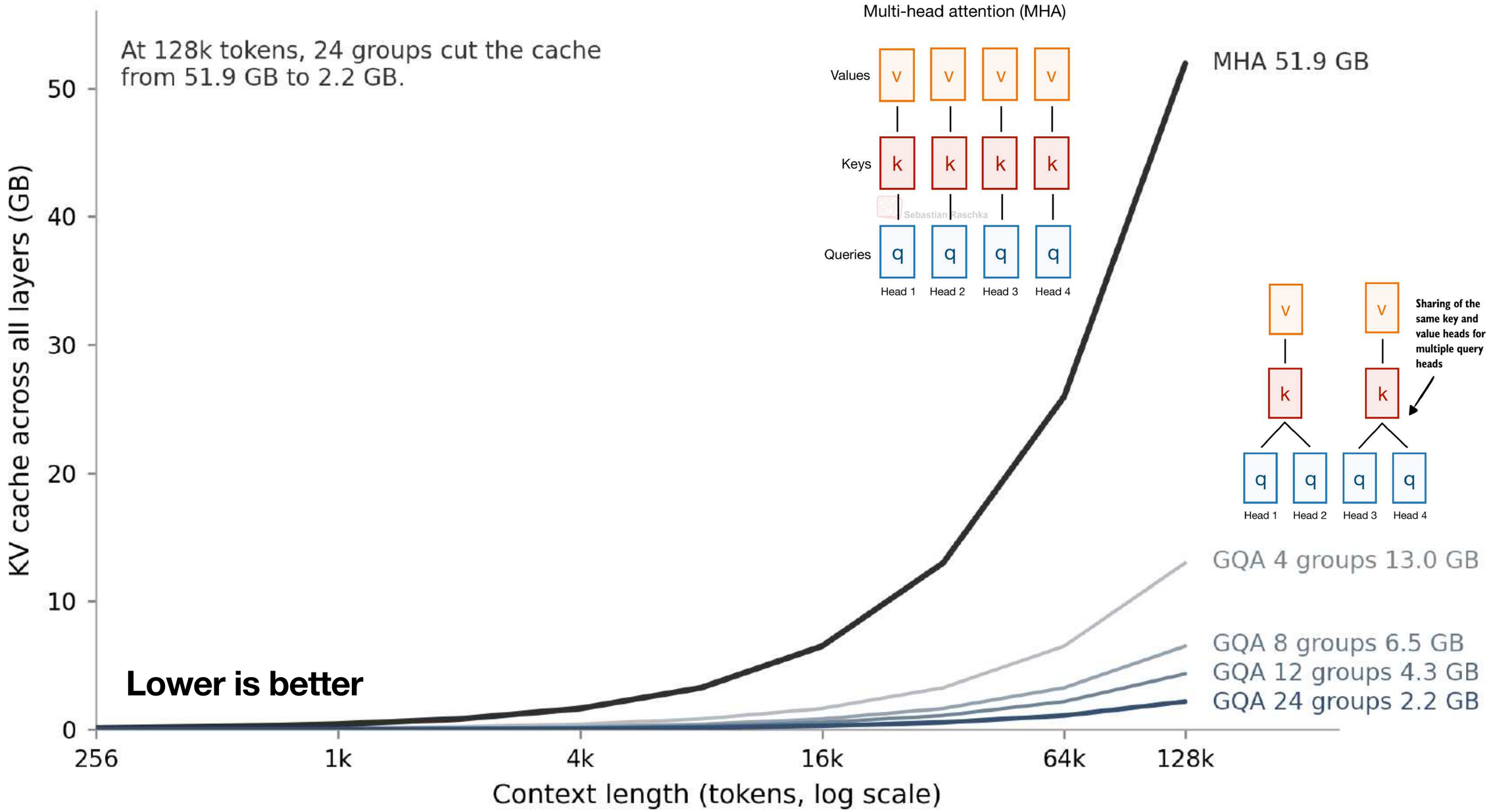




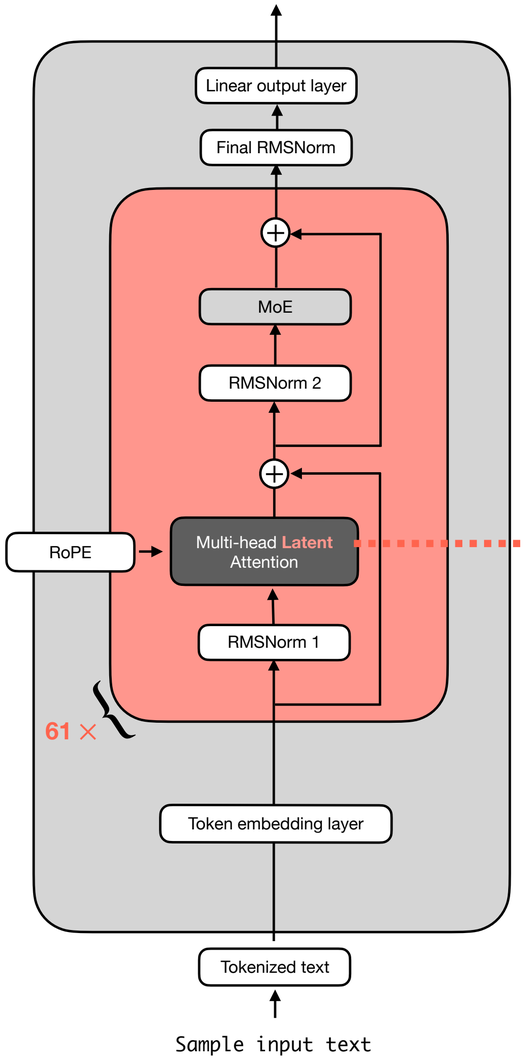
https://magazine.sebastianraschka.com/p/visual-attention-variants

GQA's shared K/V heads sharply reduce KV-cache

24 heads, 48 layers, 2048-dim model, batch 1, bf16

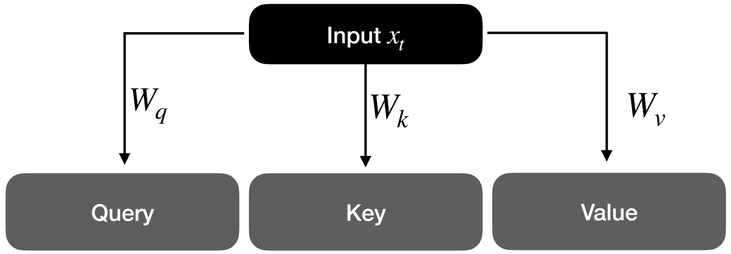


DeepSeek V3/R1



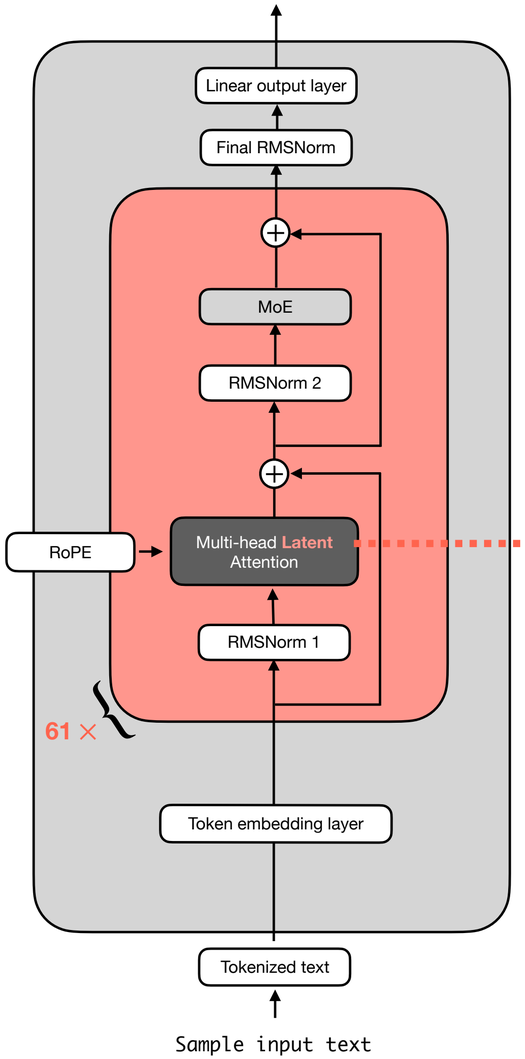
Regular Multi-Head Attention (MHA)

Inference step *t:*

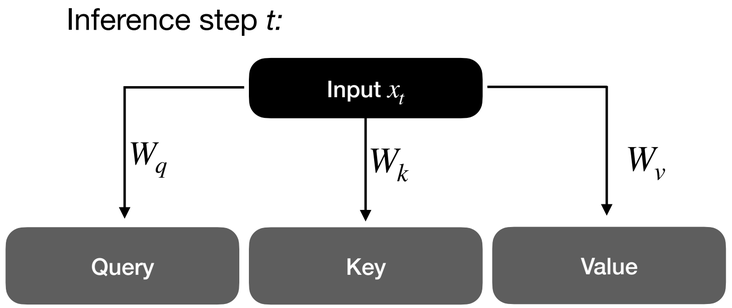


Method 2 to reduce KV cache size

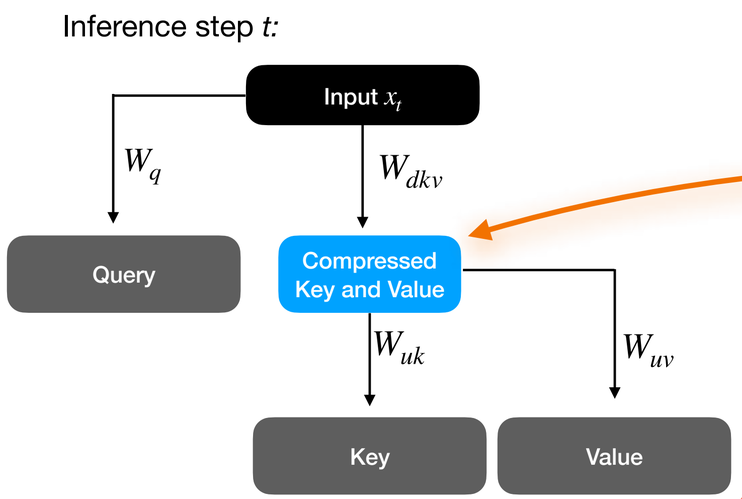
DeepSeek V3/R1



Regular Multi-Head Attention (MHA)



Multi-Head Latent Attention (MLA)



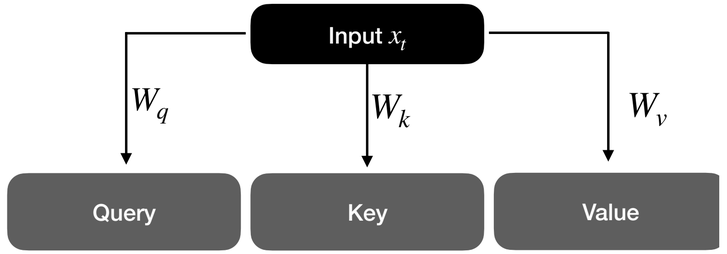
Method 2 to reduce KV cache size

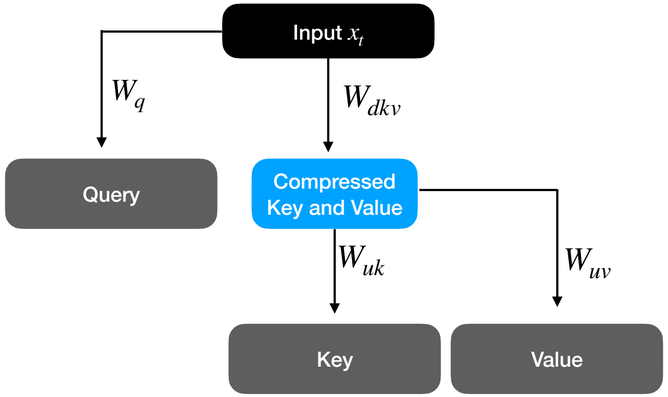
Store “small” intermediate compressed state in KV cache

MLA's latent cache sharply reduces KV-cache

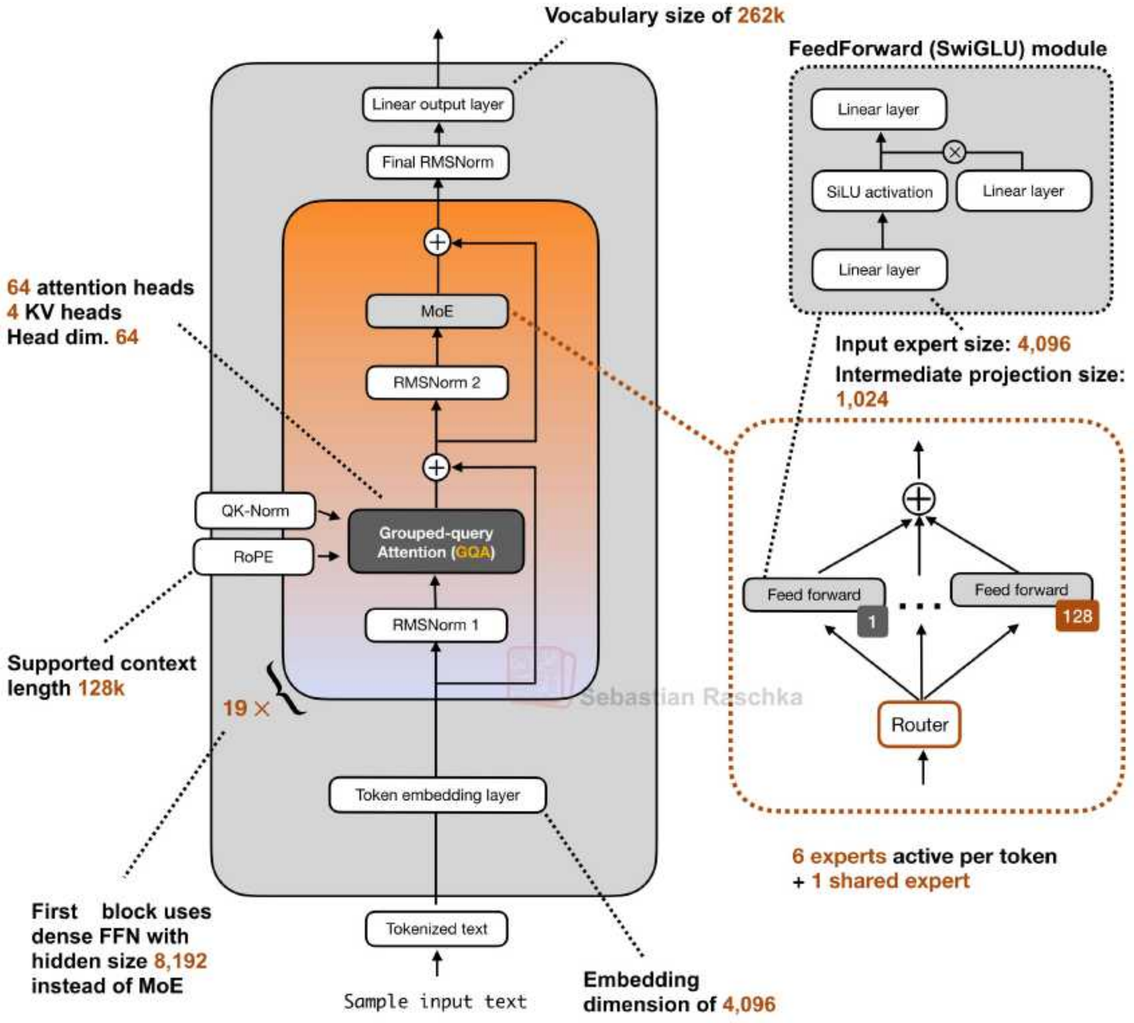
24 heads, 48 layers, 2048-dim model, batch 1, bf16

At 128k tokens, latent_dim 64 cuts the cache from 51.5 GB to 0.8 GB.

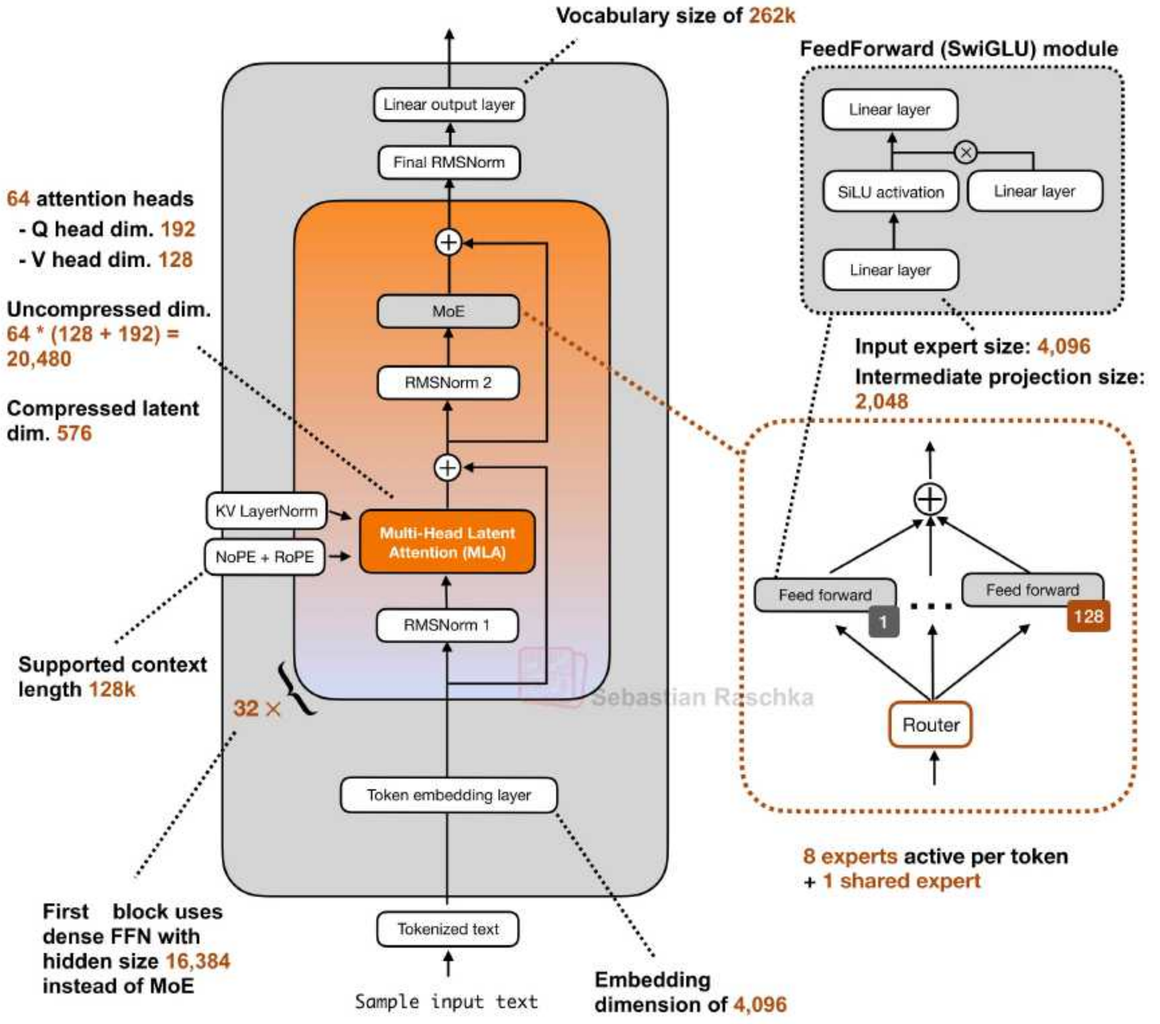




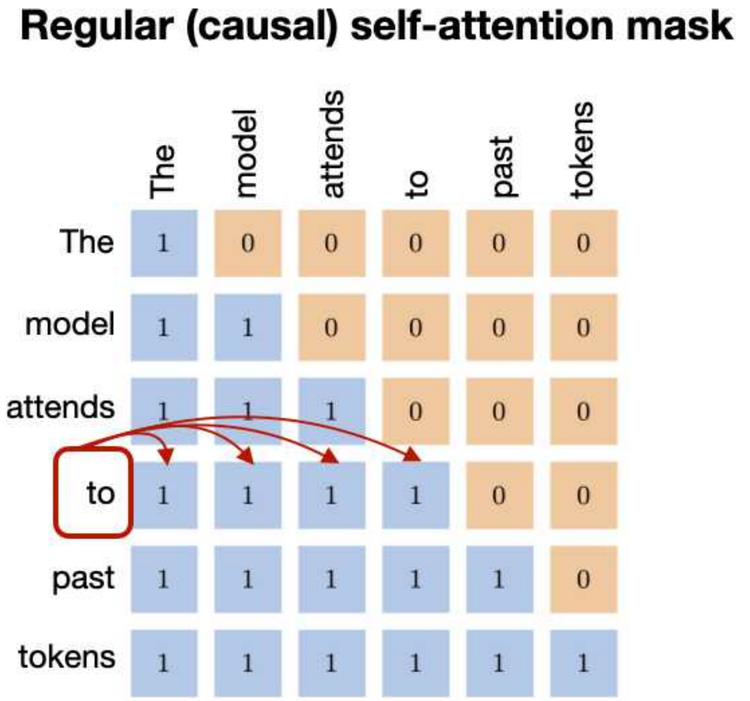
Sarvam (30B)



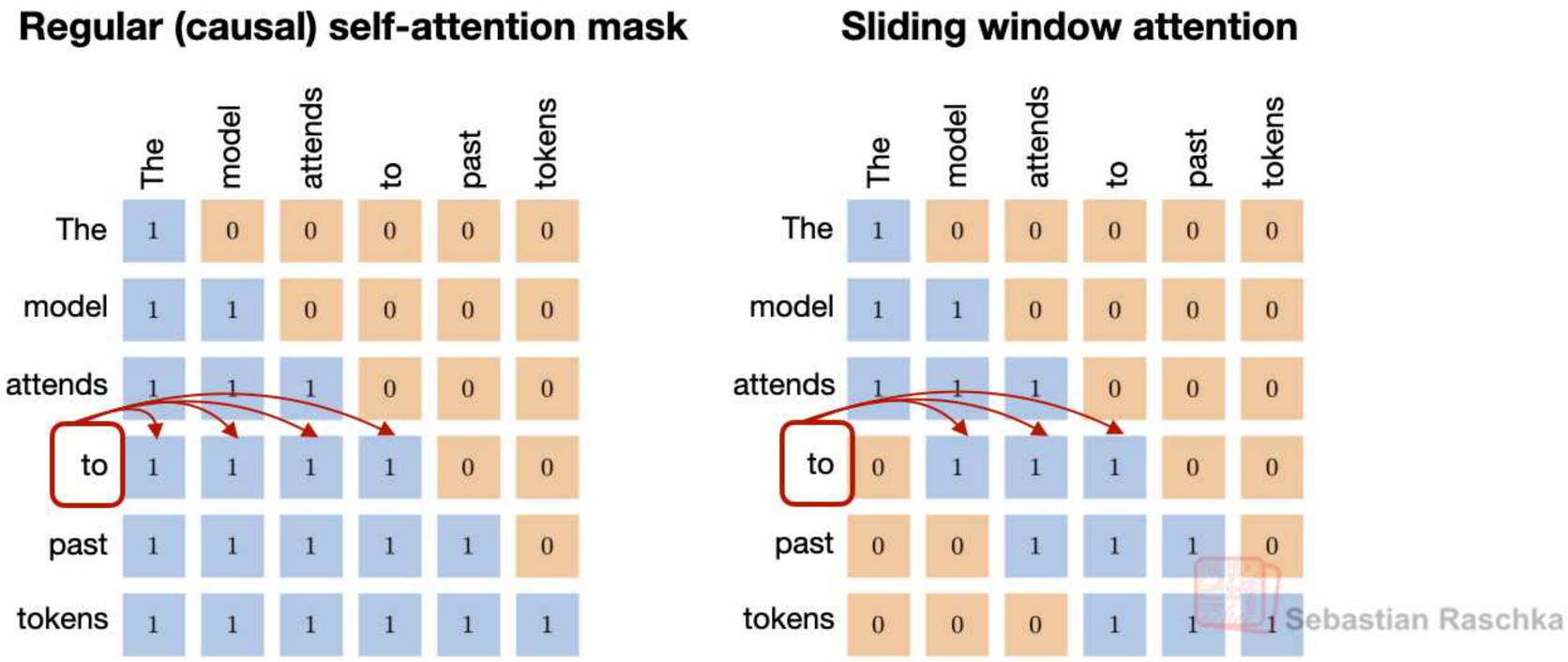
Sarvam (105B)



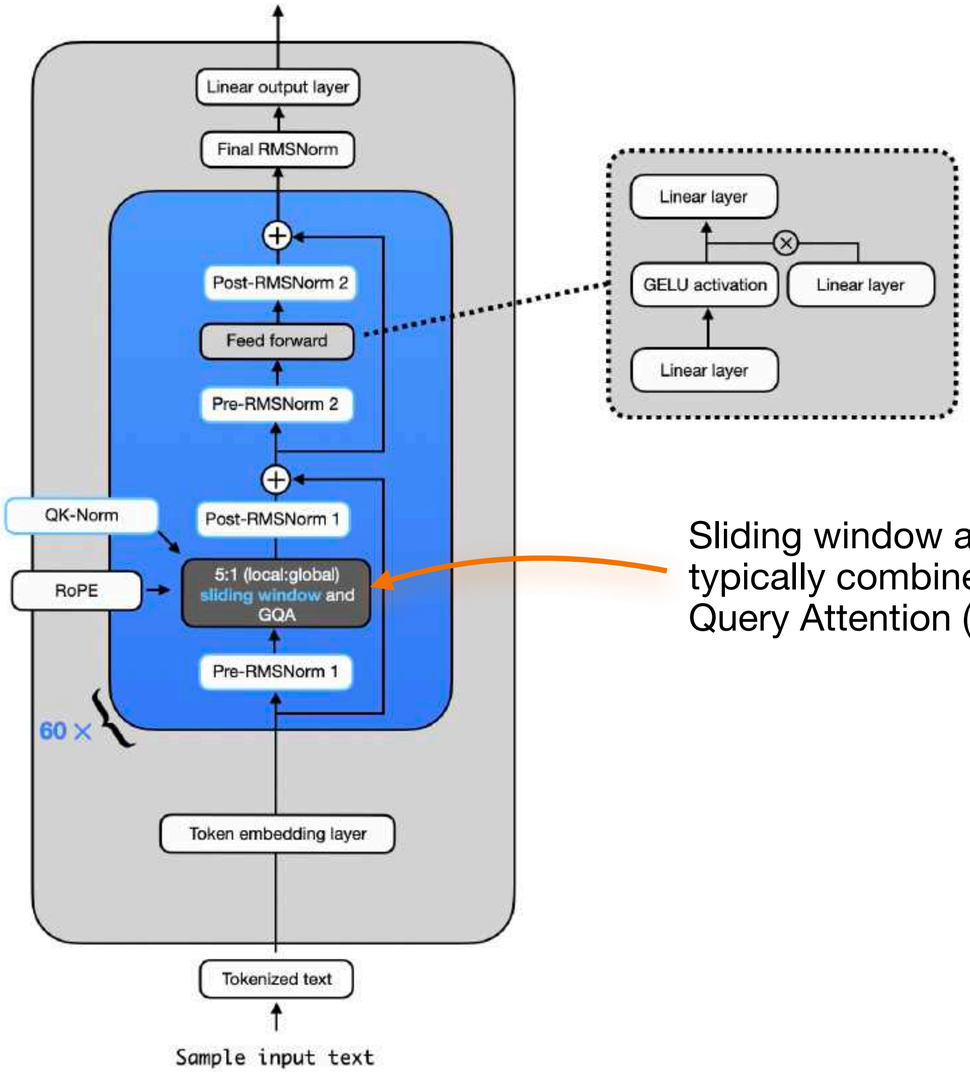
Method 3 to reduce KV cache size



Method 3 to reduce KV cache size



Gemma 4 (31B)

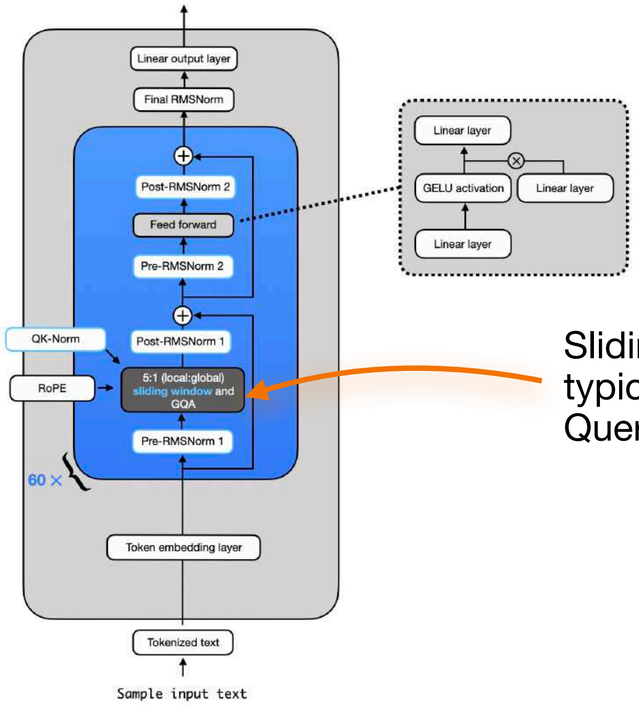


Sliding window attention (SWA) typically combined with Grouped Query Attention (GQA)

Sliding windows sharply reduce KV-cache growth

48 heads, 36 layers, 4096-dim model, batch 1, bf16; SWA ratio 5:1, window 2048

At 128k tokens, GQA 4 groups + SWA cuts the cache from 77.3 GB to 3.5 GB.



Sliding window attention (SWA) typically combined with Grouped Query Attention (GQA)

MHA 77.3 GB

GQA 4 groups 19.3 GB

MHA + SWA 13.9 GB

GQA 4 groups + SWA 3.5 GB

KV cache across all layers (GB)

0 10 20 30 40 50 60 70 80 90

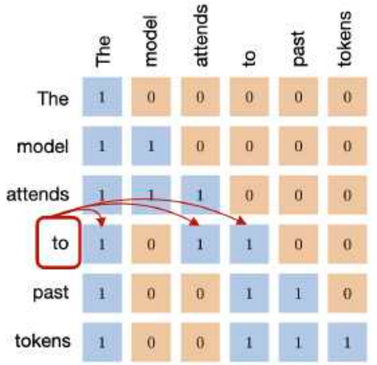
256 1k 4k 16k 64k 128k

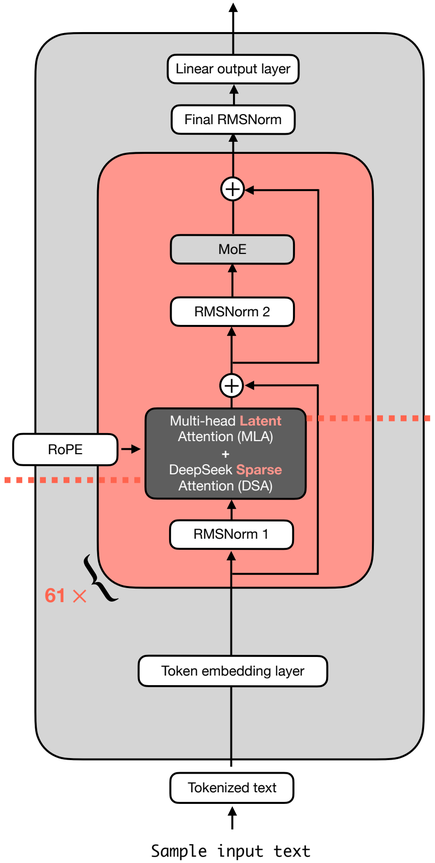
Context length (tokens, log scale)

Method 4 to reduce KV cache size

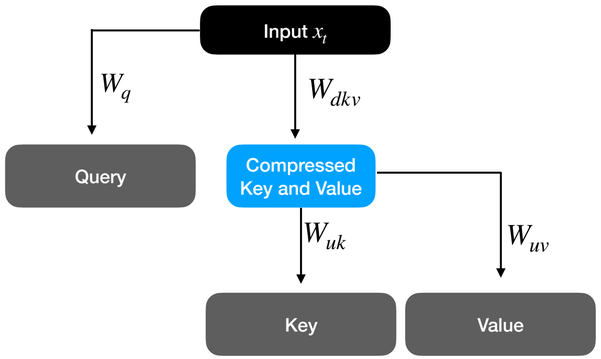
DeepSeek V3.2

DeepSeek Sparse Attention (DSA)

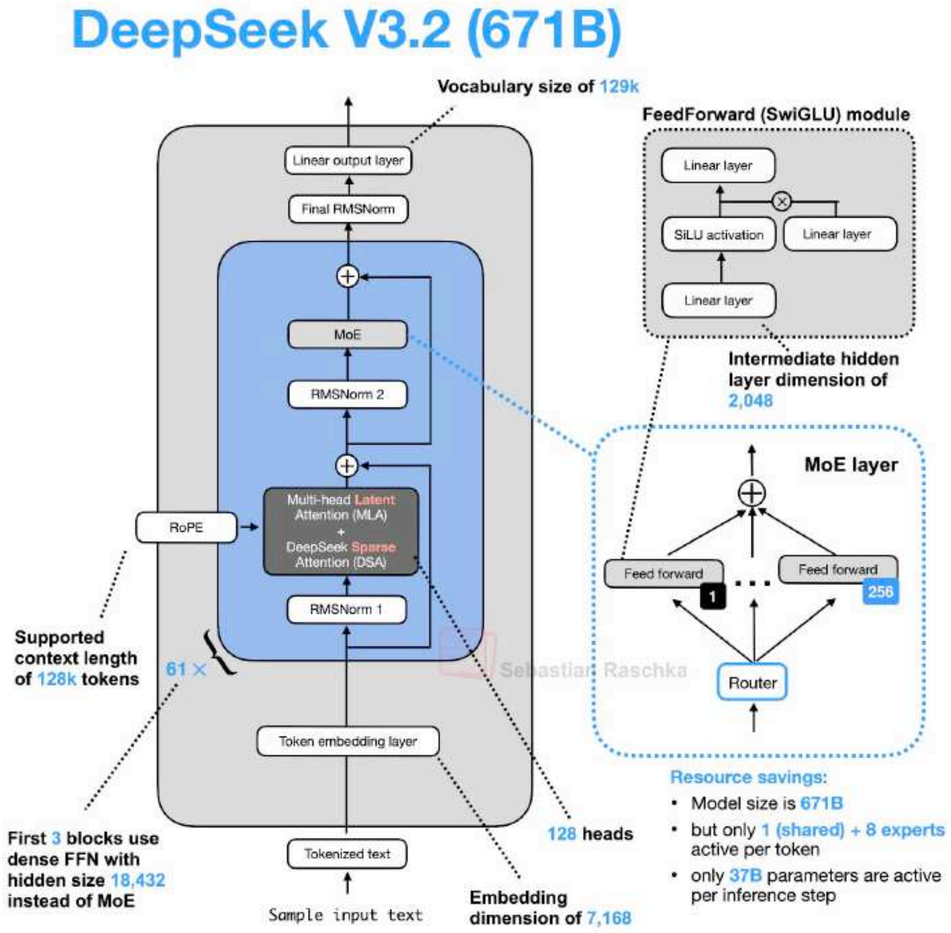


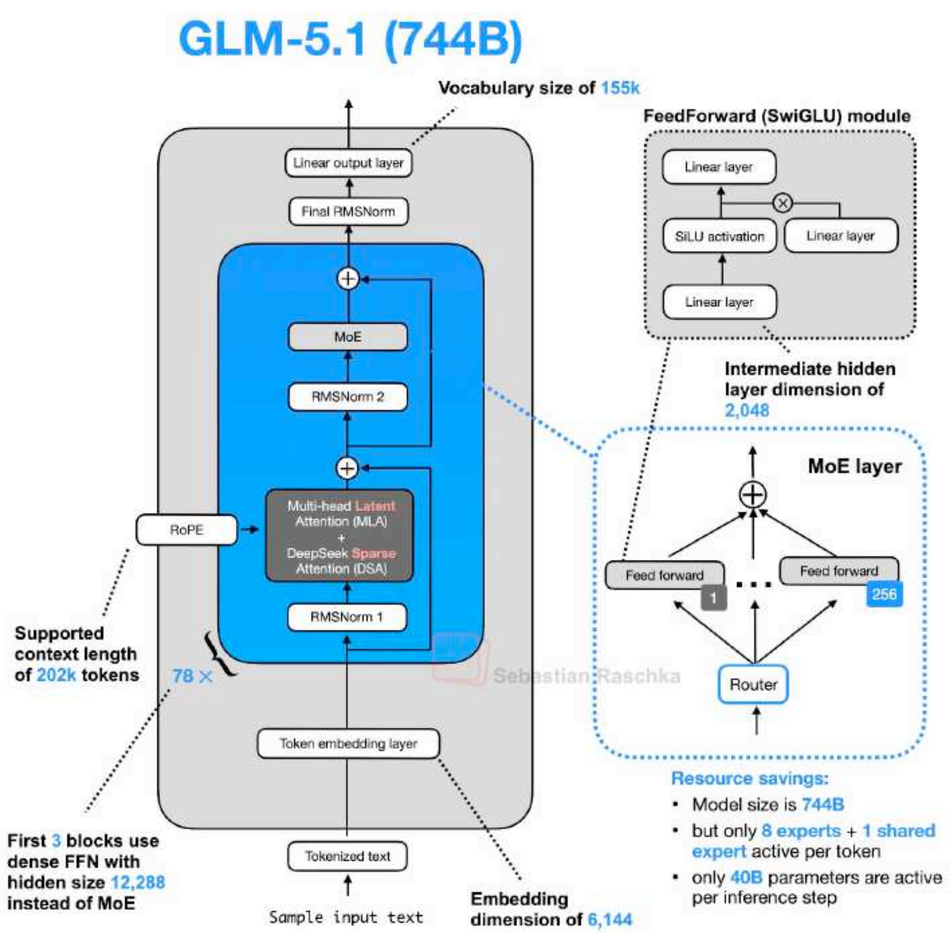


Multi-Head Latent Attention (MLA)



GLM-5.1 (strongest open-weight model) uses DeepSeek-V3.2 architecture

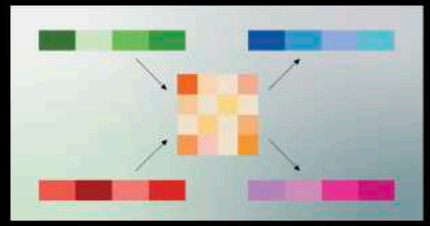




Method 5 to reduce KV cache size

Google Research

TurboQuant: Redefining AI efficiency with extreme compression



March 24, 2026 ·

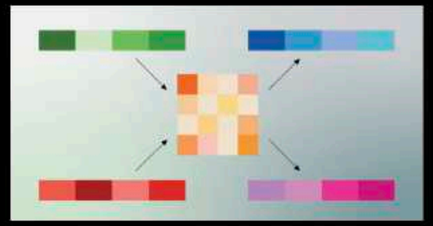
Amir Zandieh, Research Scientist, and Vahab Mirrokni, VP and Google Fellow, Google Research

https://research.google/blog/turboquant-redefining-ai-efficiency-with-extreme-compression/

Method 5 to reduce KV cache size

Google Research

TurboQuant: Redefining AI efficiency with extreme compression

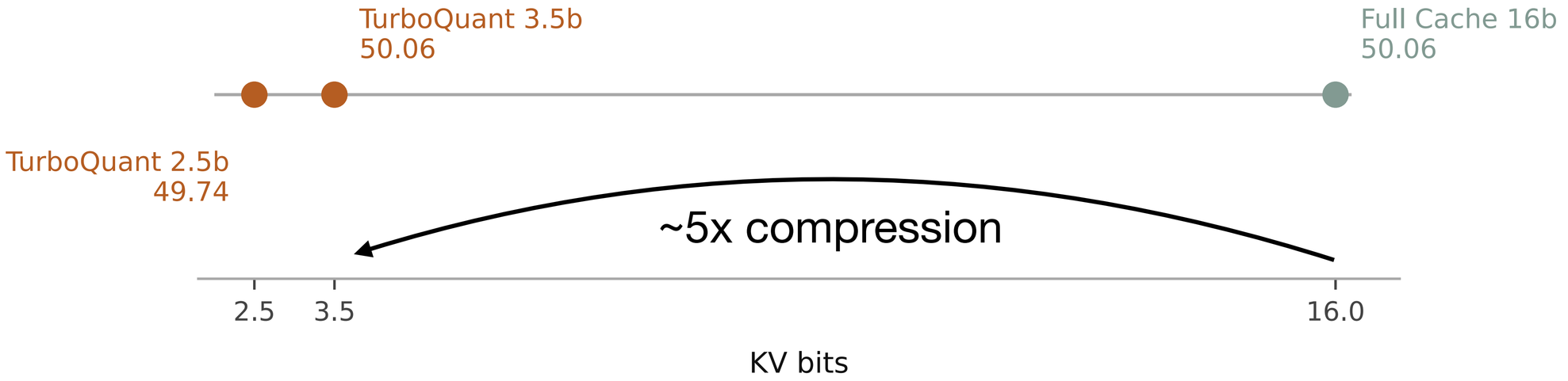


March 24, 2026 ·

Amir Zandieh, Research Scientist, and Vahab Mirrokni, VP and Google Fellow, Google Research

https://research.google/blog/turboquant-redefining-ai-efficiency-with-extreme-compression/

TurboQuant reaches 50.06 at 3.5-bit KV; Full Cache uses 16-bit KV



More methods to reduce KV cache size:

- **Linear attention in Qwen3.5, Nvidia Nemotron 3**
  https://magazine.sebastianraschka.com/p/beyond-standard-llms

- **k=v in Gemma 4**

More Things on the Horizon

Beyond architecture and training optimizations

More Things on the Horizon

Conventional LLM

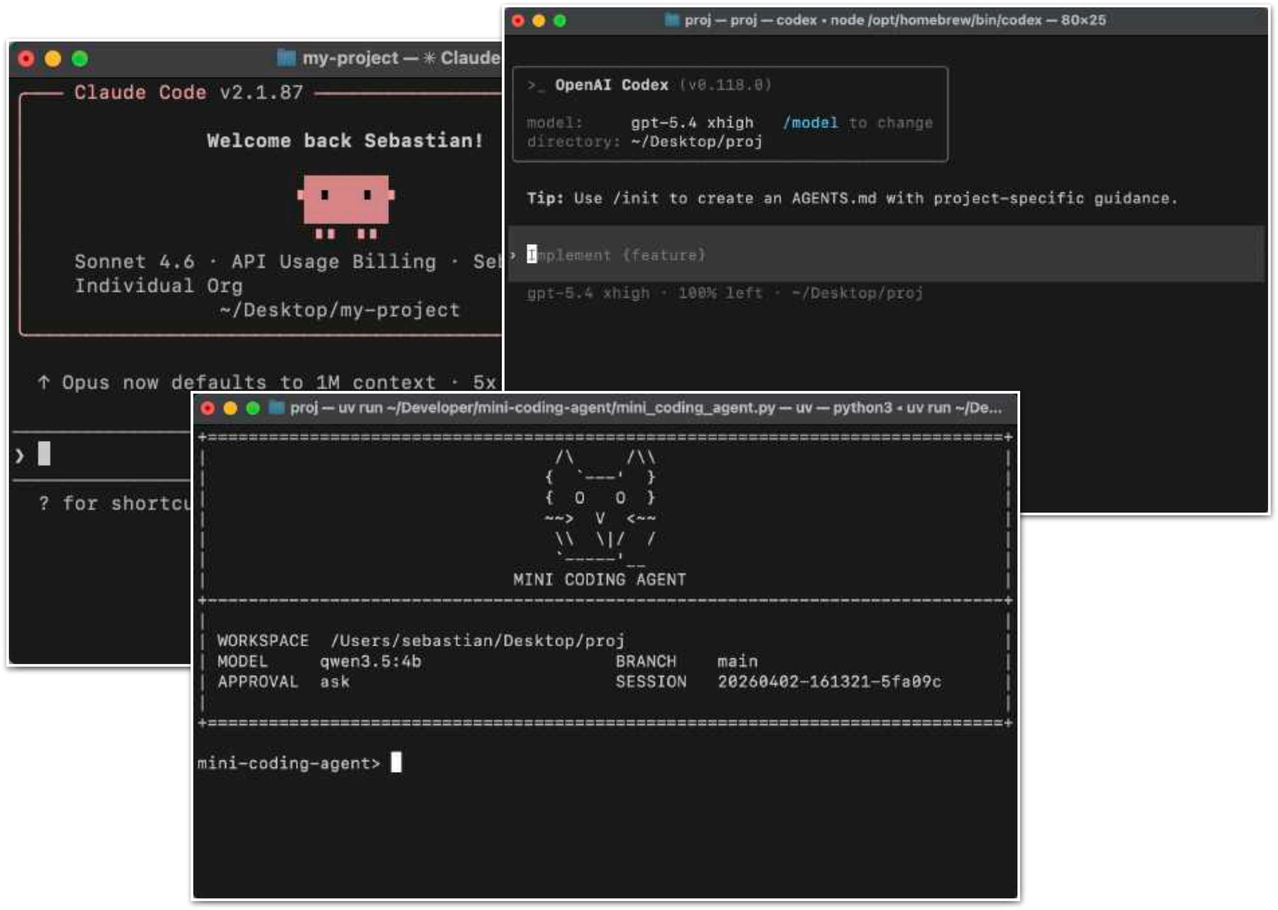
Reasoning LLM
(more expensive to run)

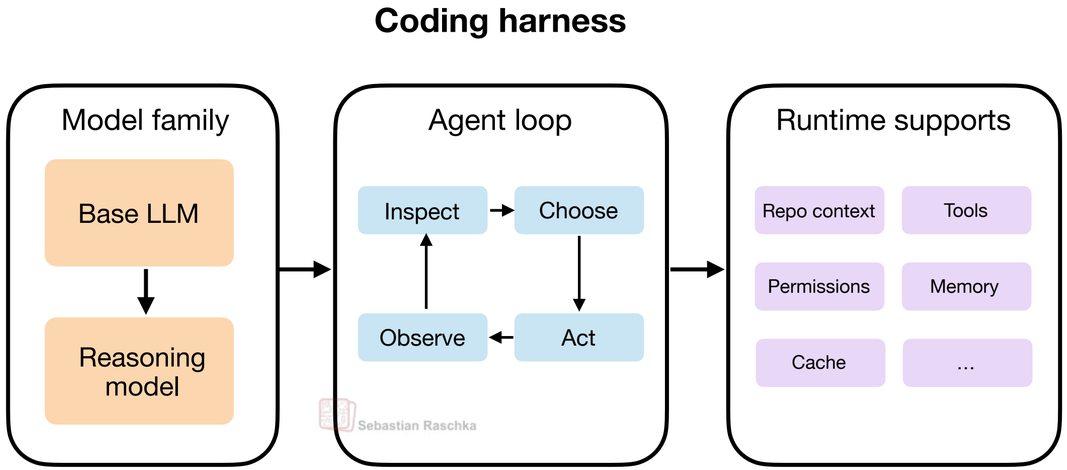
More Things on the Horizon

Conventional LLM

Reasoning LLM
(more expensive to run)

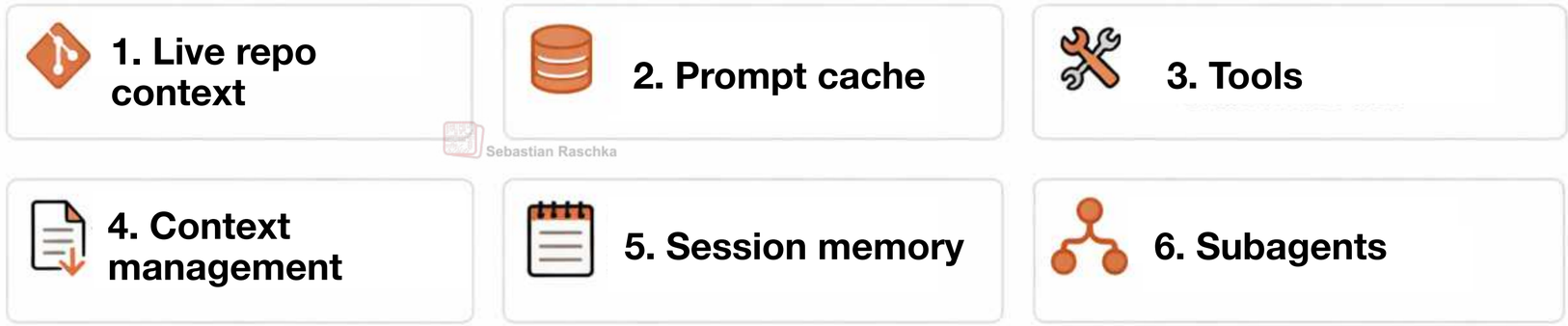
LLM in agent harness





https://magazine.sebastianraschka.com/p/components-of-a-coding-agent

Components of a coding agent



https://github.com/rasbt/mini-coding-agent

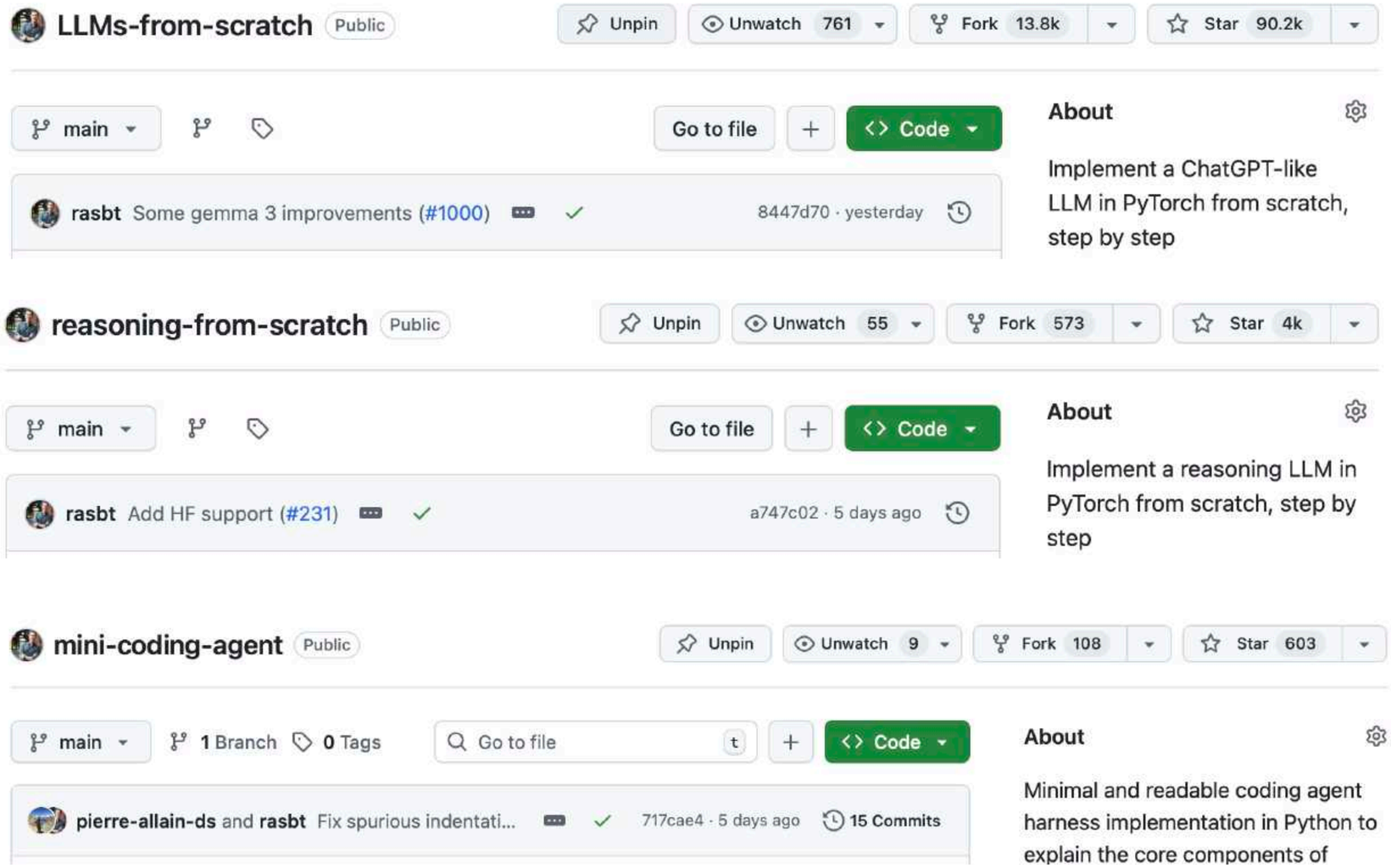
“Getting started” tips

Beginner

Understanding how
LLMs work

Implementing a few
LLMs and training algos
from scratch

Beginner: Understand how things work



“Getting started” tips

Beginner

Understanding how LLMs work

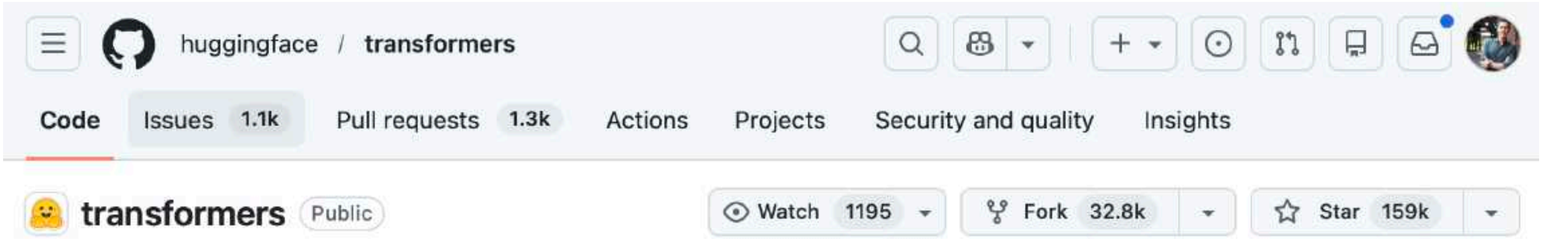
Implementing a few LLMs and training algos from scratch

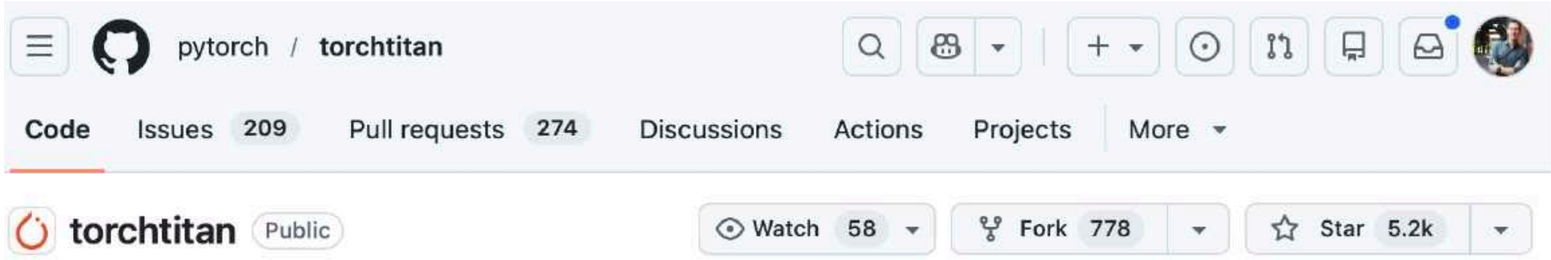
Intermediate

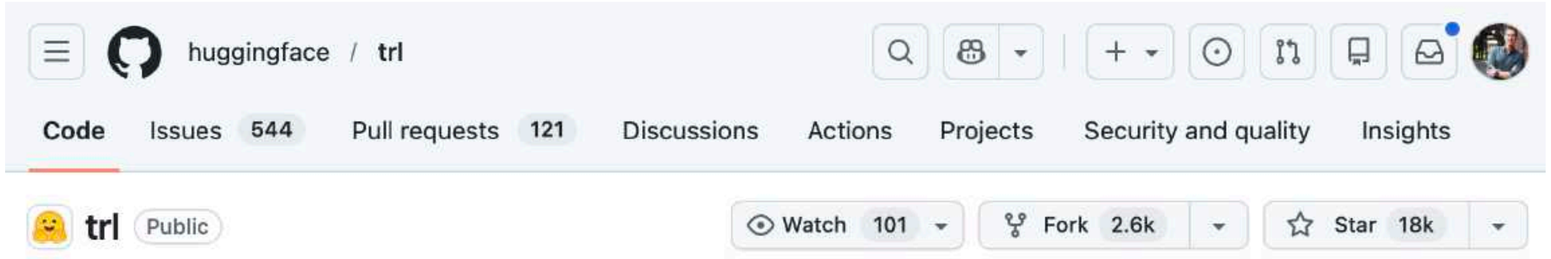
Wrapper libraries to access more complex algos

Scale to small set of GPUs

Intermediate: Wrappers that make advanced features easier to use







“Getting started” tips

Beginner

Understanding how LLMs work

Implementing a few LLMs and training algos from scratch

Intermediate

Wrapper libraries to access more complex algos

Scale to small set of GPUs

Pro

Larger-scale, professional training runs

Inspect and adapt code bases used by teams that trained popular LLMs successfully

Pro: Project and production code bases

*most training stacks are not public

Allen AI's OLMo 3 (32B)

https://github.com/allenai/olmo-core

https://github.com/allenai/open-instruct

Hugging Face's SmolLM3 (3B)

https://github.com/huggingface/nanotron/tree/smollm3

https://github.com/huggingface/trl

Intellect-3 (106B MoE)

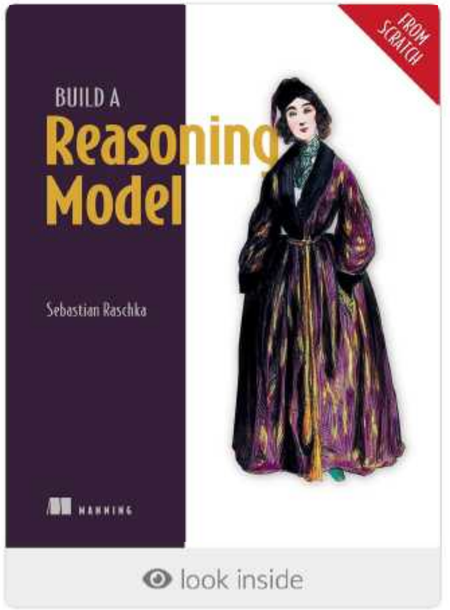
https://github.com/PrimeIntellect-ai/prime-rl

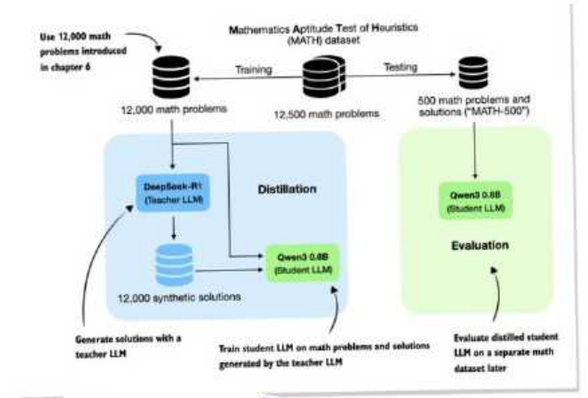
Nemotron 3

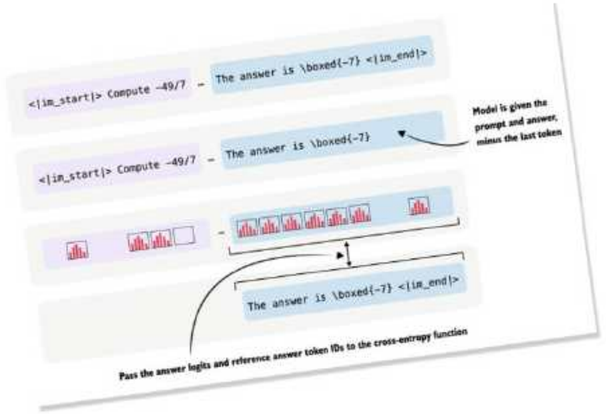
Nano (30B MoE) & Super (120B MoE)

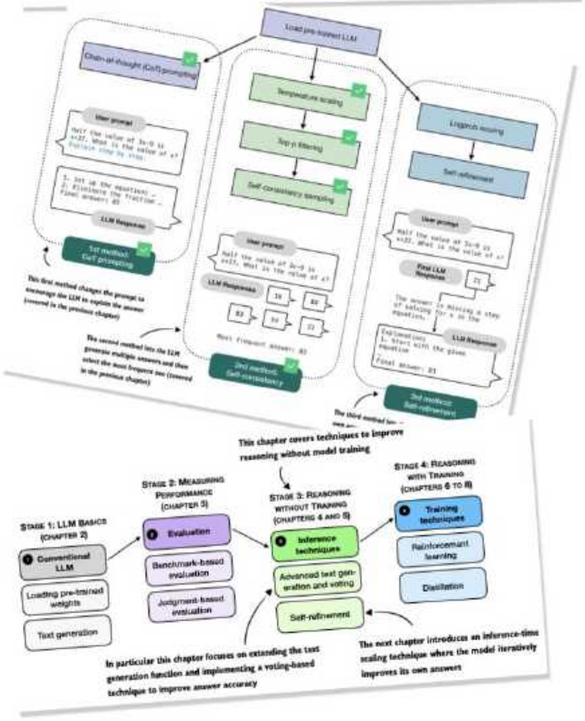
https://github.com/NVIDIA-NeMo/Nemotron

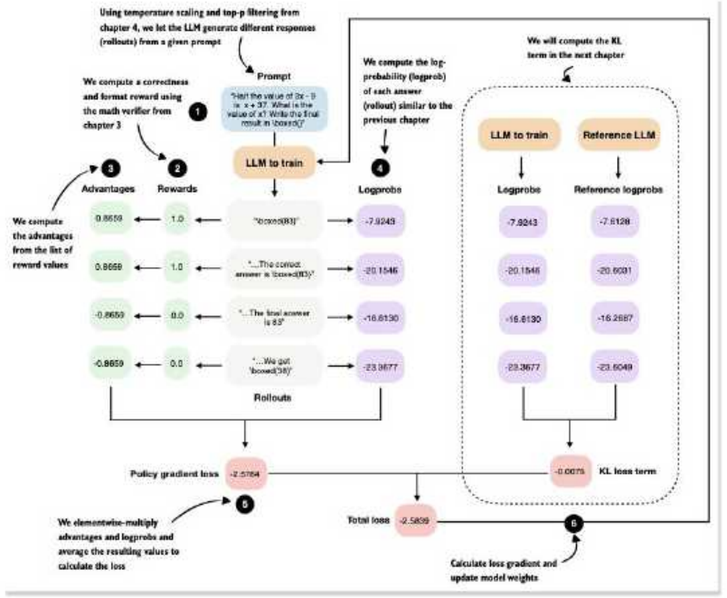
https://github.com/NVIDIA/Megatron-LM











- Amazon (pre-order), https://amzn.to/4aAKiFY
- Manning, https://mng.bz/Nwr7 (complete book in early access, pre-final layout, 528 pages)

Book signing at 16:00