

RAG, Agents, Finetuning, frameworks

Konrad Handrick

Workshop 3: stepping stone in AI
Dezember 2025

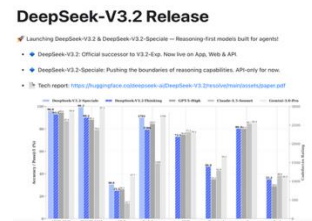
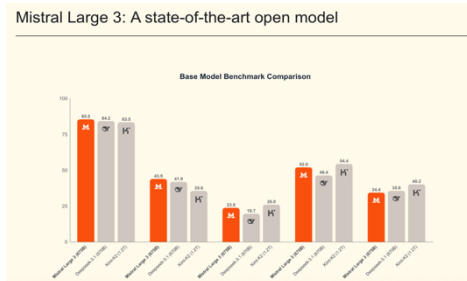
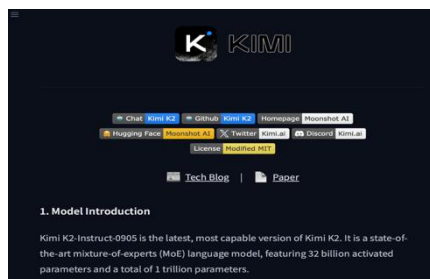
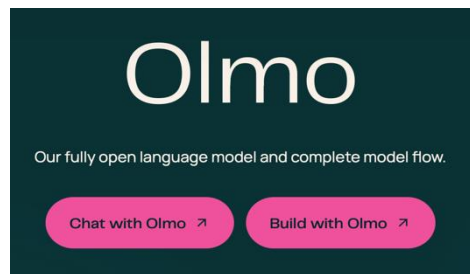
Today's menu (less theoretical)



1. Catching up a little
2. Quantization
3. Finetuning
4. RAG
5. Agents

... it'll become a little more complex though!

A few news first



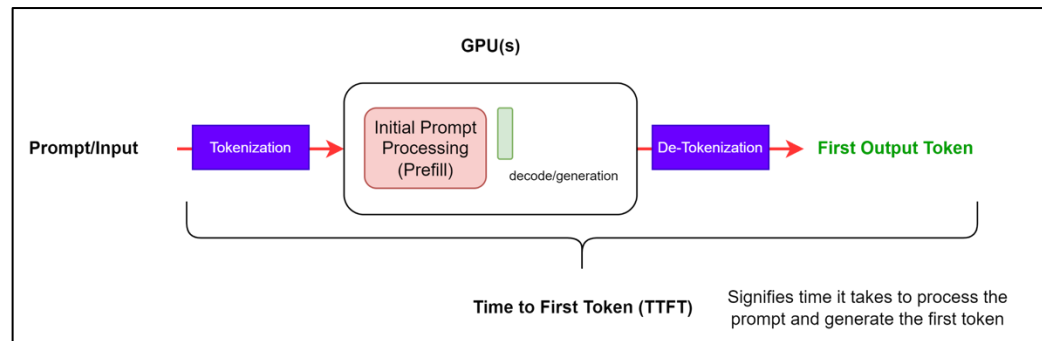
Top left: <https://allenai.org/olmo>

Top right: <https://mistral.ai/news/mistral-3/>

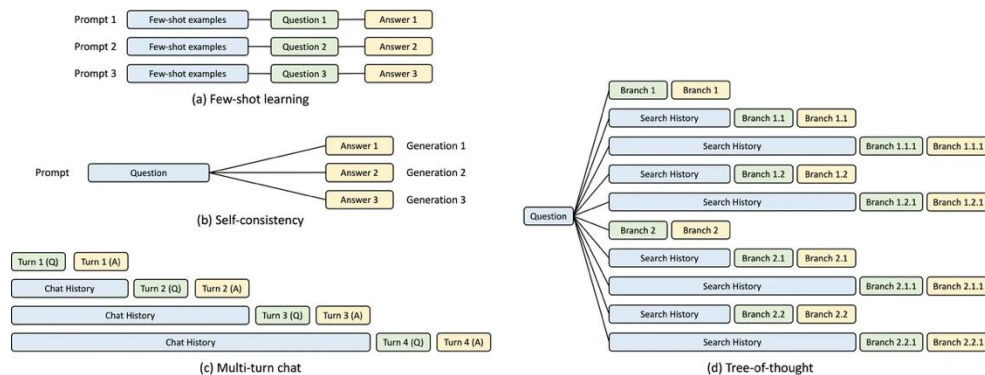
Bottom left: <https://api-docs.deepseek.com/news/news251201>

Bottom right: <https://huggingface.co/moonshotai/Kimi-K2-Instruct-0905>

Addendum: KV Caches



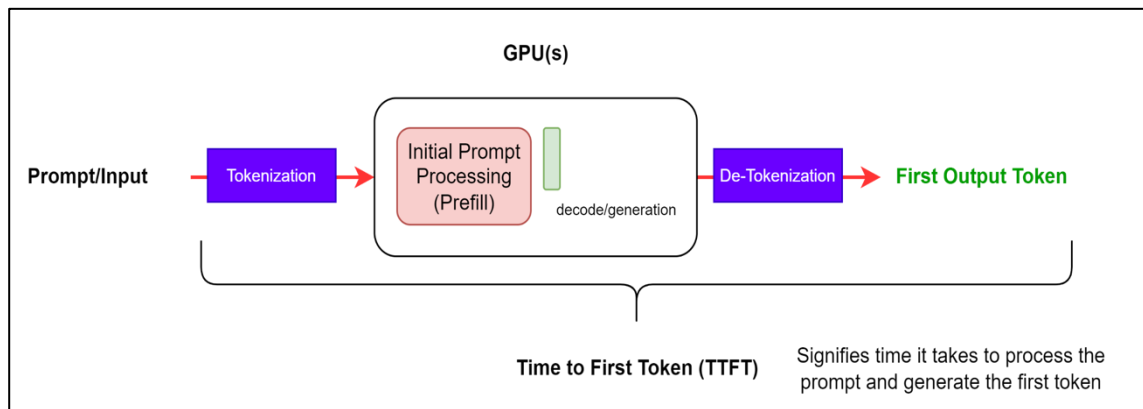
Language redundancy, prefixes,
positionally-encoded tokens



Top: <https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html>

Right: <https://arxiv.org/abs/2309.06180>

Addendum: KV Caches



Top: <https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html>

Right: <https://sankalp.bearblog.dev/how-prompt-caching-works/>

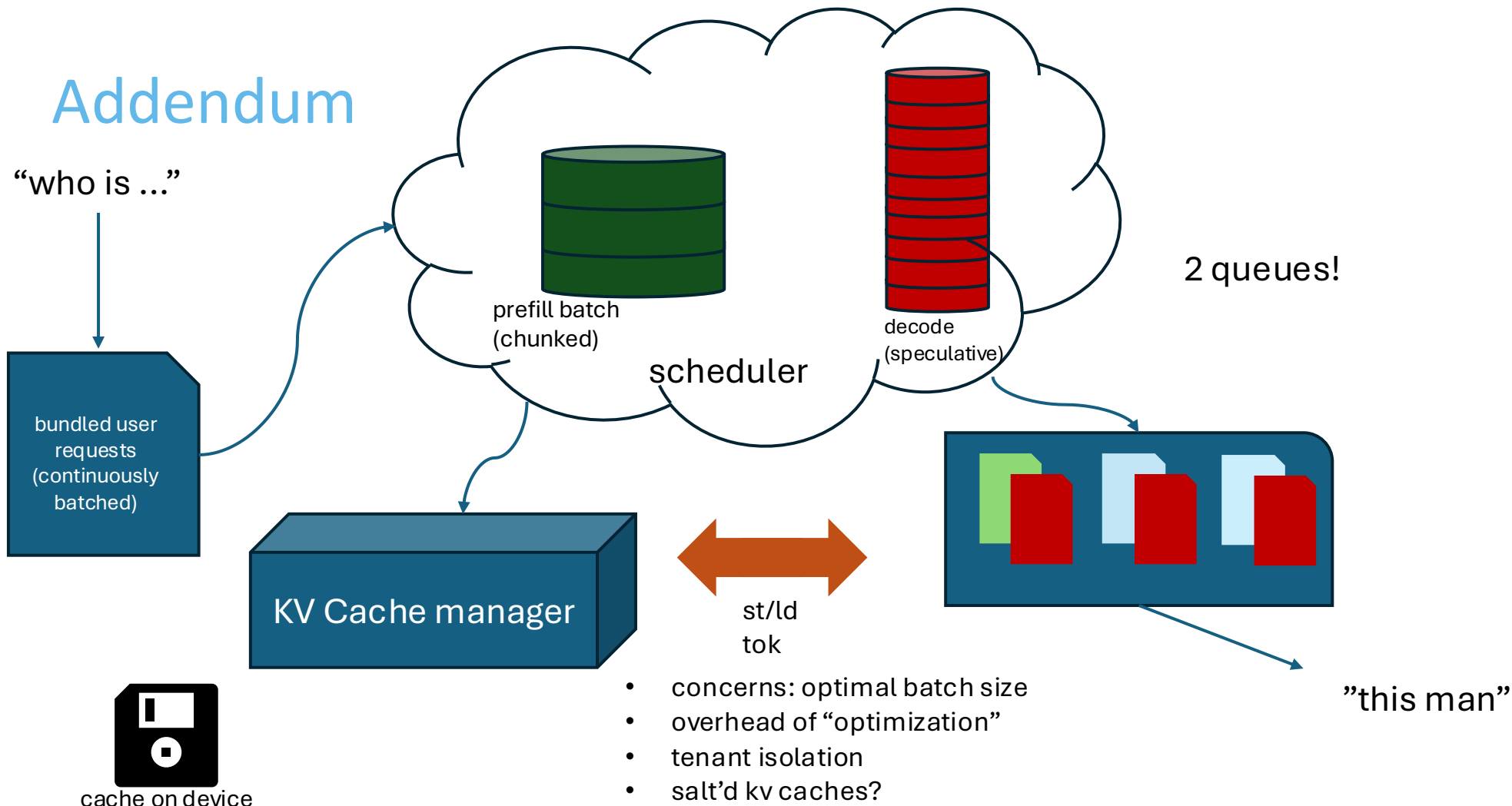
t=0: REQUEST 1	t=1: REQUEST 2																																								
"You are an expert in LLM inference optimization. You understand KV caching, paged attention, and memory management. Be concise. How does prefix caching reduce time to first token?"	"You are an expert in LLM inference optimization. You understand KV caching, paged attention, and memory management. Be concise. What is the block size in VLLM?"																																								
(80 tokens → 5 blocks)	(80 tokens → 5 blocks)																																								
HASH	HASH																																								
h0 = 0xA1 h1 = 0xB2 h2 = 0xC3 h3 = 0xD4 h4 = 0xE5	h0, h1, h2 = 0xA1, 0xB2, 0xC3 - same h3 = 0xF6, h4 = 0xG7 - different																																								
CACHE LOOKUP	CACHE LOOKUP																																								
cache: { empty }	cache: { 0xA1-phy_0, 0xB2-phy_1, ... }																																								
all MISS	0xA1-HIT 0xB2-HIT 0xC3-HIT 0xF6-MISS																																								
longest_hit = 0 blocks	longest_hit = 3 blocks (48 tokens)																																								
ALLOCATE & BLOCK TABLE	ALLOCATE & BLOCK TABLE																																								
free: [0,1,2...] → pop 5	free: [5,6,7...] → pop 2																																								
<table><tr><td>Blk0</td><td>Blk1</td><td>Blk2</td><td>Blk3</td><td>Blk4</td></tr><tr><td>phy_0</td><td>phy_1</td><td>phy_2</td><td>phy_3</td><td>phy_4</td></tr><tr><td>ref:1</td><td>ref:1</td><td>ref:1</td><td>ref:1</td><td>ref:1</td></tr><tr><td>new</td><td>new</td><td>new</td><td>new</td><td>new</td></tr></table>	Blk0	Blk1	Blk2	Blk3	Blk4	phy_0	phy_1	phy_2	phy_3	phy_4	ref:1	ref:1	ref:1	ref:1	ref:1	new	new	new	new	new	<table><tr><td>Blk0</td><td>Blk1</td><td>Blk2</td><td>Blk3</td><td>Blk4</td></tr><tr><td>phy_0</td><td>phy_1</td><td>phy_2</td><td>phy_5</td><td>phy_6</td></tr><tr><td>ref:2</td><td>ref:2</td><td>ref:2</td><td>ref:1</td><td>ref:1</td></tr><tr><td>reuse</td><td>reuse</td><td>reuse</td><td>new</td><td>new</td></tr></table>	Blk0	Blk1	Blk2	Blk3	Blk4	phy_0	phy_1	phy_2	phy_5	phy_6	ref:2	ref:2	ref:2	ref:1	ref:1	reuse	reuse	reuse	new	new
Blk0	Blk1	Blk2	Blk3	Blk4																																					
phy_0	phy_1	phy_2	phy_3	phy_4																																					
ref:1	ref:1	ref:1	ref:1	ref:1																																					
new	new	new	new	new																																					
Blk0	Blk1	Blk2	Blk3	Blk4																																					
phy_0	phy_1	phy_2	phy_5	phy_6																																					
ref:2	ref:2	ref:2	ref:1	ref:1																																					
reuse	reuse	reuse	new	new																																					
PREFILL	PREFILL																																								
COMPUTE COMPUTE COMPUTE COMPUTE COMPUTE	SKIP SKIP SKIP COMPUTE COMPUTE																																								
total: 80 tokens	total: 32 tokens (saved 48!)																																								

system prompt

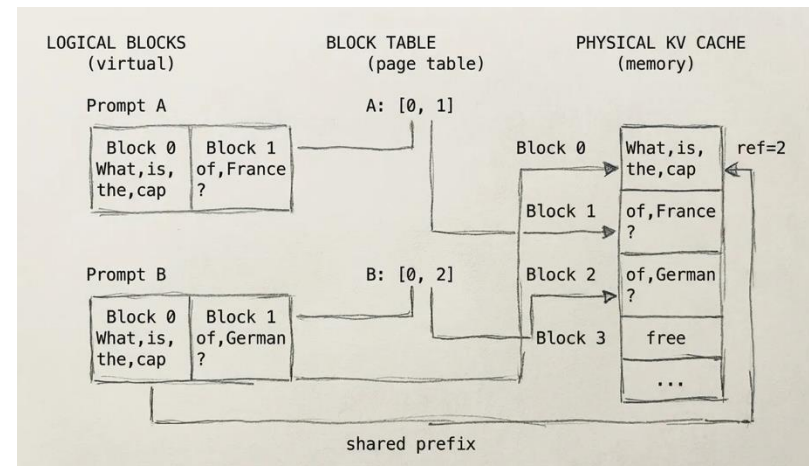
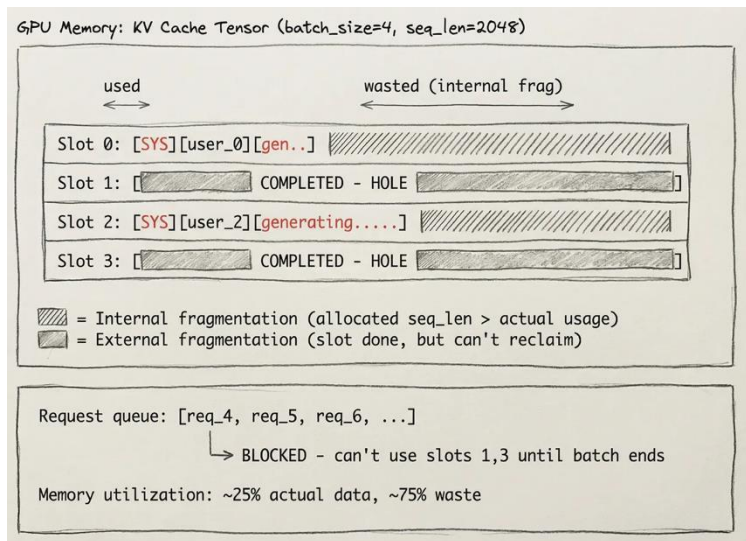
cache eviction

t=0: REQUEST 1	t=1: REQUEST 2																																								
"You are an expert in LLM inference optimization. You understand KV caching, paged attention, and memory management. Be concise. How does prefix caching reduce time to first token?"	"You are an expert in LLM inference optimization. You understand KV caching, paged attention, and memory management. Be concise. What is the block size in VLLM?"																																								
(80 tokens → 5 blocks)	(80 tokens → 5 blocks)																																								
HASH	HASH																																								
h0 = 0xA1 h1 = 0xB2 h2 = 0xC3 h3 = 0xD4 h4 = 0xE5	h0, h1, h2 = 0xA1, 0xB2, 0xC3 ← same h3 = 0xF6, h4 = 0xG7 ← different																																								
CACHE LOOKUP	CACHE LOOKUP																																								
cache: { empty }	cache: { 0xA1-phy_0, 0xB2-phy_1, ... }																																								
all MISS	0xA1-HIT 0xB2-HIT 0xC3-HIT 0xF6-MISS																																								
longest_hit = 0 blocks	longest_hit = 3 blocks (48 tokens)																																								
ALLOCATE & BLOCK TABLE	ALLOCATE & BLOCK TABLE																																								
free: [0,1,2...] → pop 5	free: [5,6,7...] → pop 2																																								
<table><tr><td>Blk0</td><td>Blk1</td><td>Blk2</td><td>Blk3</td><td>Blk4</td></tr><tr><td>phy_0</td><td>phy_1</td><td>phy_2</td><td>phy_3</td><td>phy_4</td></tr><tr><td>ref:1</td><td>ref:1</td><td>ref:1</td><td>ref:1</td><td>ref:1</td></tr><tr><td>new</td><td>new</td><td>new</td><td>new</td><td>new</td></tr></table>	Blk0	Blk1	Blk2	Blk3	Blk4	phy_0	phy_1	phy_2	phy_3	phy_4	ref:1	ref:1	ref:1	ref:1	ref:1	new	new	new	new	new	<table><tr><td>Blk0</td><td>Blk1</td><td>Blk2</td><td>Blk3</td><td>Blk4</td></tr><tr><td>phy_0</td><td>phy_1</td><td>phy_2</td><td>phy_5</td><td>phy_6</td></tr><tr><td>ref:2</td><td>ref:2</td><td>ref:2</td><td>ref:1</td><td>ref:1</td></tr><tr><td>reuse</td><td>reuse</td><td>reuse</td><td>new</td><td>new</td></tr></table>	Blk0	Blk1	Blk2	Blk3	Blk4	phy_0	phy_1	phy_2	phy_5	phy_6	ref:2	ref:2	ref:2	ref:1	ref:1	reuse	reuse	reuse	new	new
Blk0	Blk1	Blk2	Blk3	Blk4																																					
phy_0	phy_1	phy_2	phy_3	phy_4																																					
ref:1	ref:1	ref:1	ref:1	ref:1																																					
new	new	new	new	new																																					
Blk0	Blk1	Blk2	Blk3	Blk4																																					
phy_0	phy_1	phy_2	phy_5	phy_6																																					
ref:2	ref:2	ref:2	ref:1	ref:1																																					
reuse	reuse	reuse	new	new																																					
PREFILL	PREFILL																																								
COMPUTE COMPUTE COMPUTE COMPUTE COMPUTE	SKIP SKIP SKIP COMPUTE COMPUTE																																								
total: 80 tokens	total: 32 tokens (saved 48!)																																								

Addendum

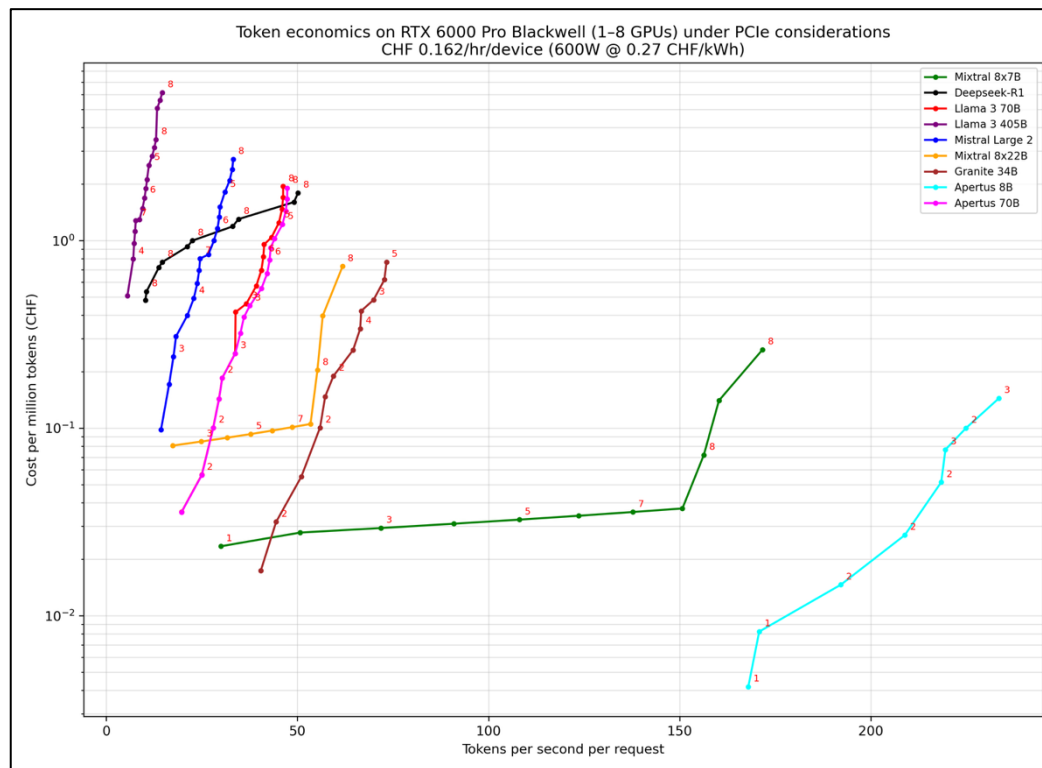


Addendum: KV Caches



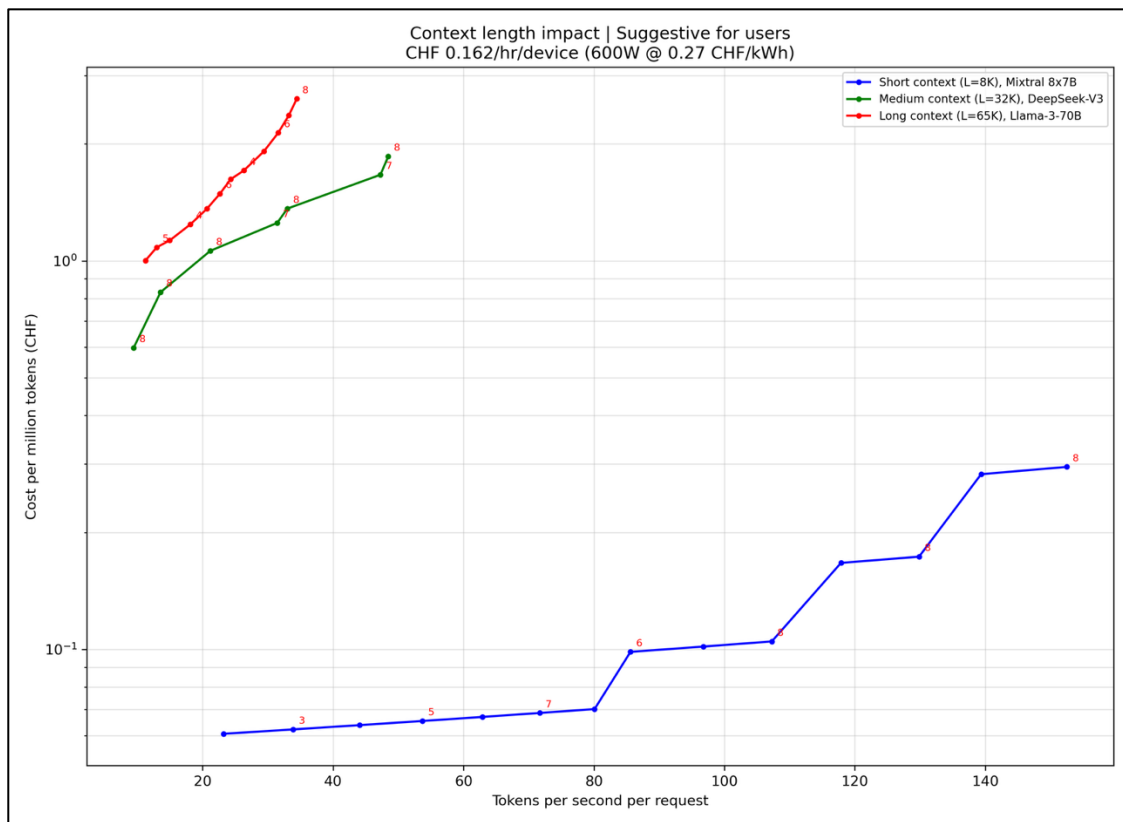
Left and right: <https://sankalp.bearblog.dev/how-prompt-caching-works/>

Pareto frontiers



Pareto frontiers

repo



The Economics of Large Language Models: Token Allocation, Fine-Tuning, and Optimal Pricing*

Dirk Bergemann[†] Alessandro Bonatti[‡] Alex Smolin[§]

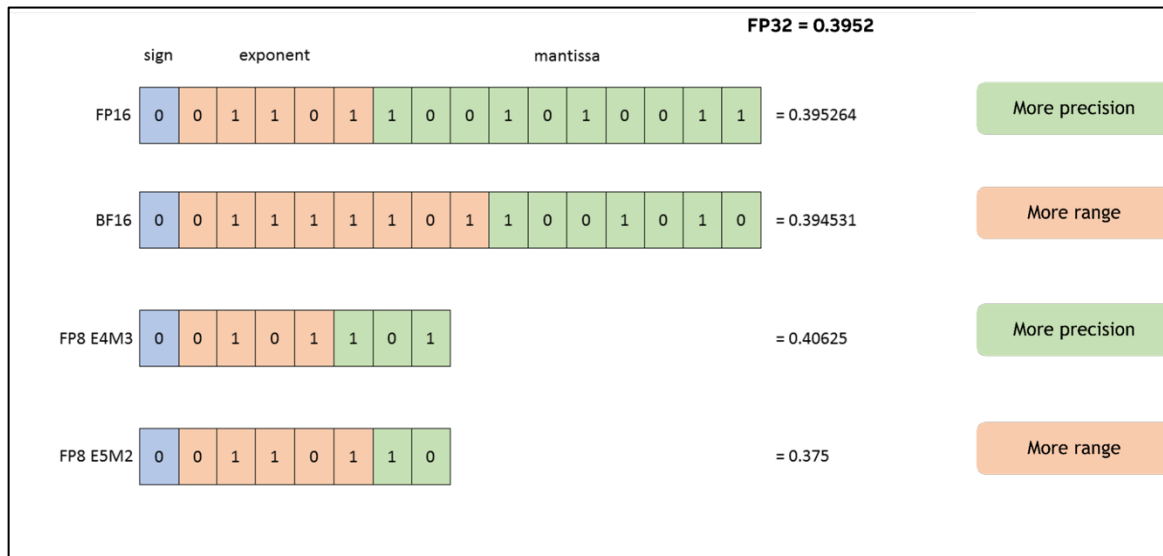
February 12, 2025

INFERENCE ECONOMICS OF LANGUAGE MODELS

Ege Erdil
Epoch
ege@epoch.ai

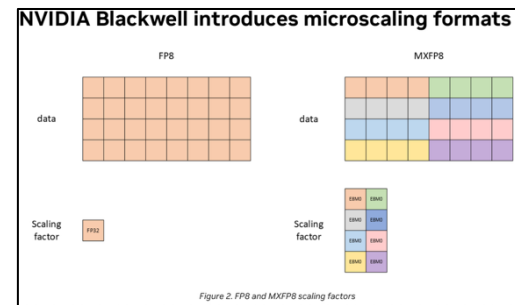
Top: <https://arxiv.org/abs/2502.07736>
Bottom: <https://arxiv.org/abs/2506.04645>

Quantization

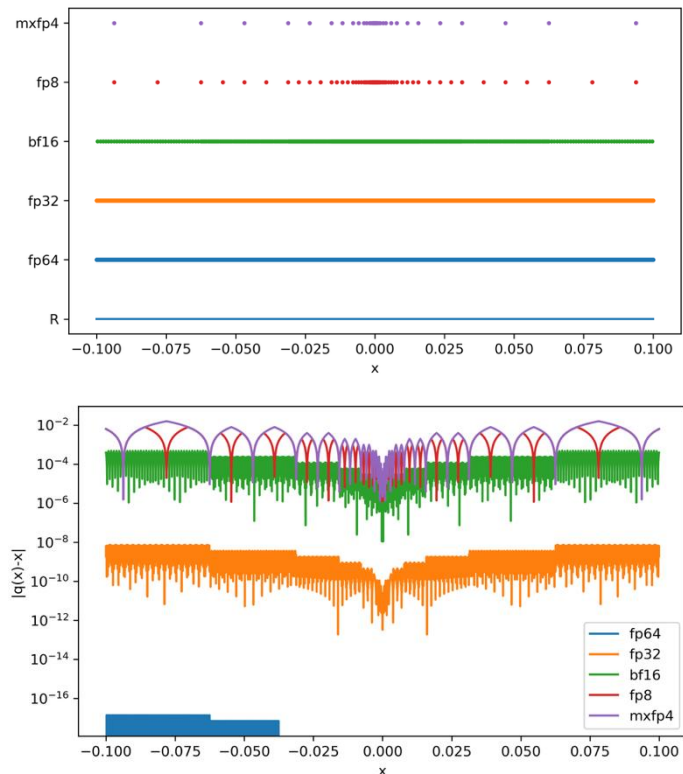
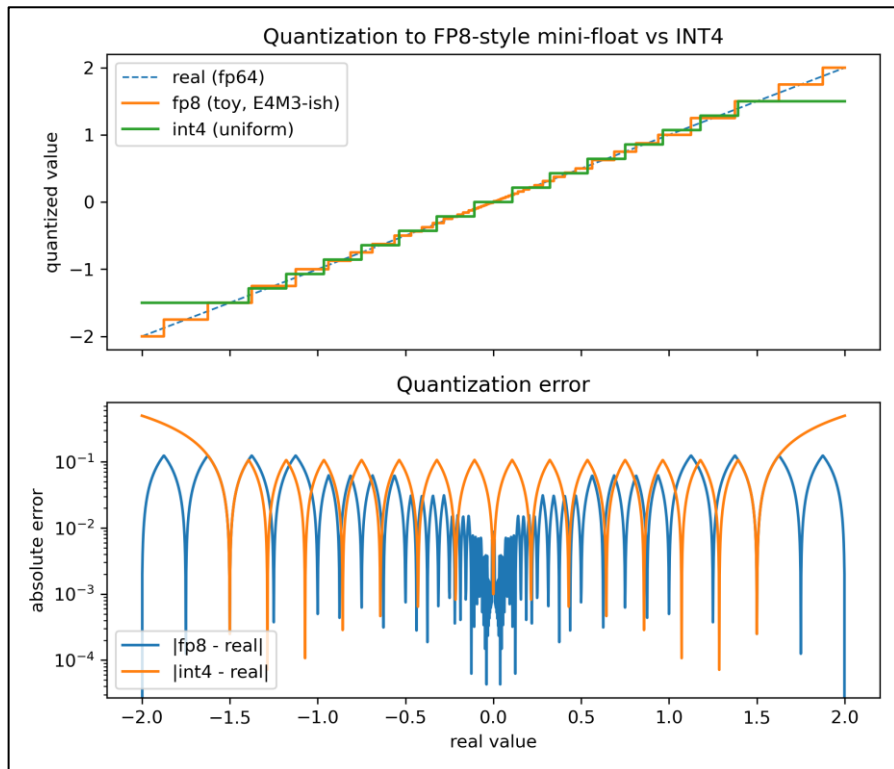


model size = num_bytes_per_weight * num_params
+ extra

Top and right: <https://developer.nvidia.com/blog/floating-point-8-an-introduction-to-efficient-lower-precision-ai-training>



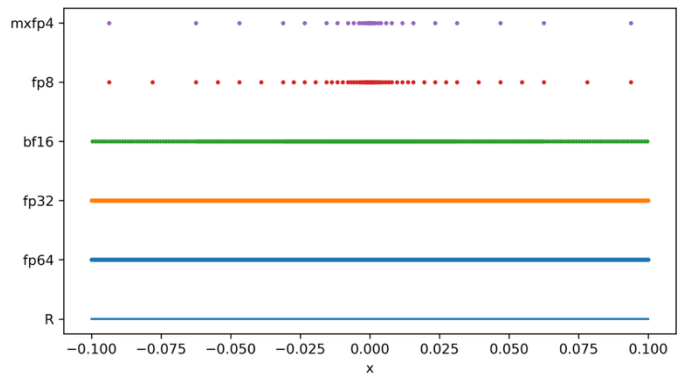
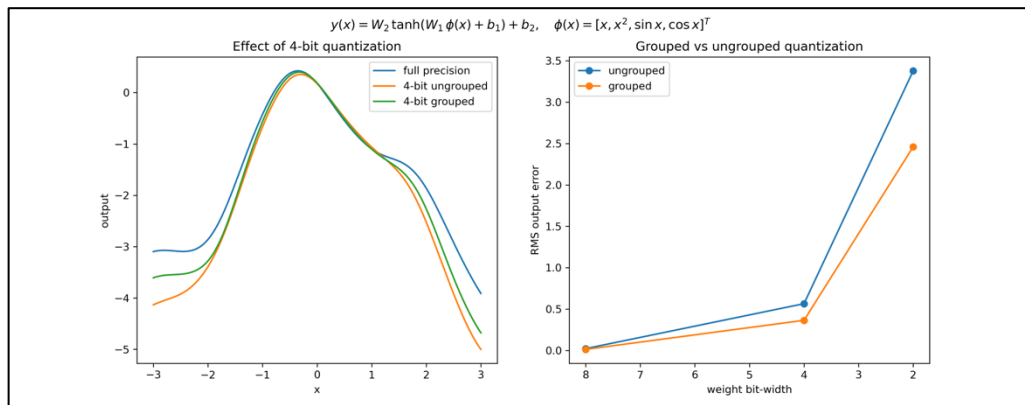
Quantization



Quantization

model size = num_bytes_per_weight * num_params
+ extra

we're on vectorized machines and want to save space
-> use groupings and quantize along



NVIDIA Blackwell introduces microscaling formats

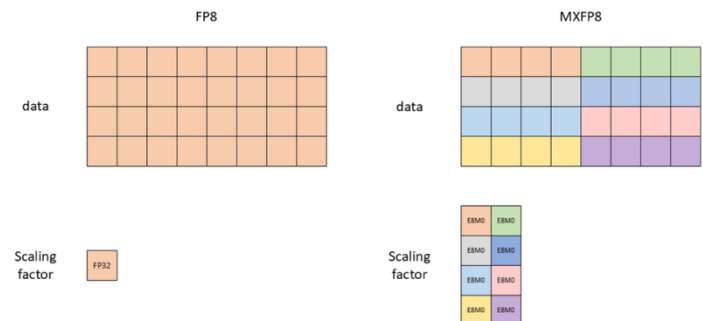


Figure 2: FP8 and MXFP8 scaling factors

Quantization

model size = num_bytes_per_weight * num_params
+ extra

training often in fp16

→ weights in bf16

→ sucks for us when we
need to serve

deepseek uses fp8 for training
+ inference

ChatGPT 5.1 Thinking ▾ Share

can you say anything about at what precision/quantization AI research companies serve their models? deepseek, anthropic, openai?

Plausible picture for OpenAI serving

Given:

- NVIDIA's inference stack (TensorRT-LLM) is heavily optimized for **FP8 and INT8** for LLMs,
NVIDIA GitHub +1
- OpenAI clearly embraces aggressive quantization for GPT-OSS,

...it's very likely that for large proprietary models OpenAI uses some mix of:

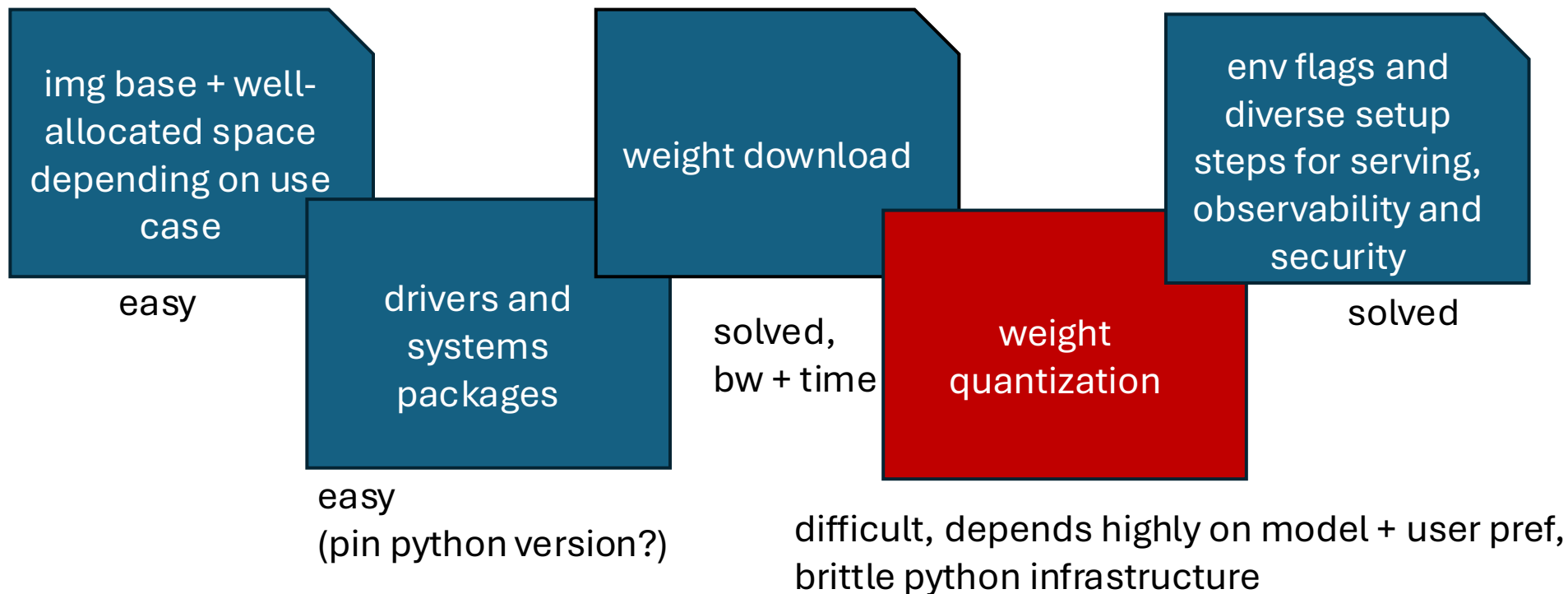
- **Weights:** FP8 / INT8 / custom FP4-like formats (possibly weight-only quantization for most layers)
- **Activations / compute:** BF16 or FP8
- **KV cache:** FP8 or FP16

But again, that's inference from hardware + ecosystem, not an official spec.

<https://chat.com>

Recap

What our images should do:



Context length usage

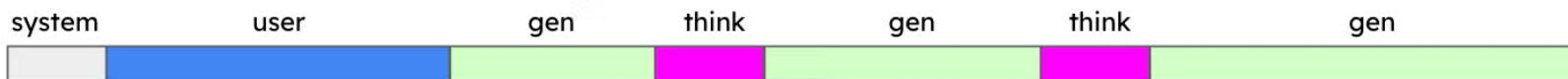
RAG Pattern



Agentic Pattern

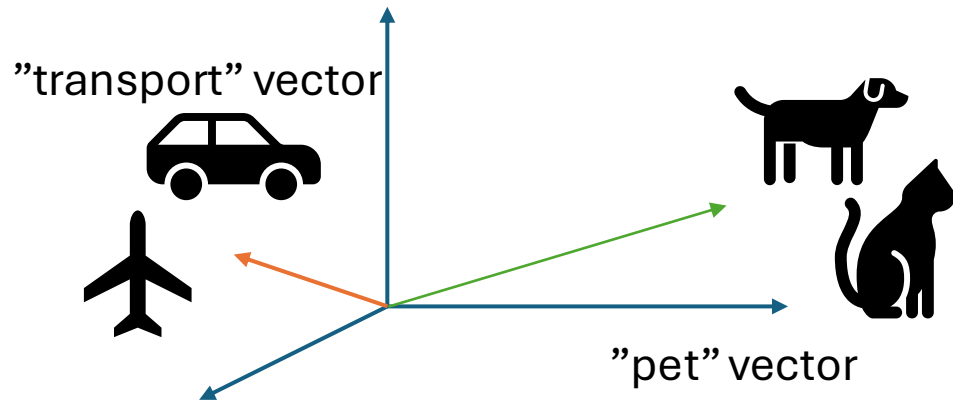


Thinking/Reasoning Pattern



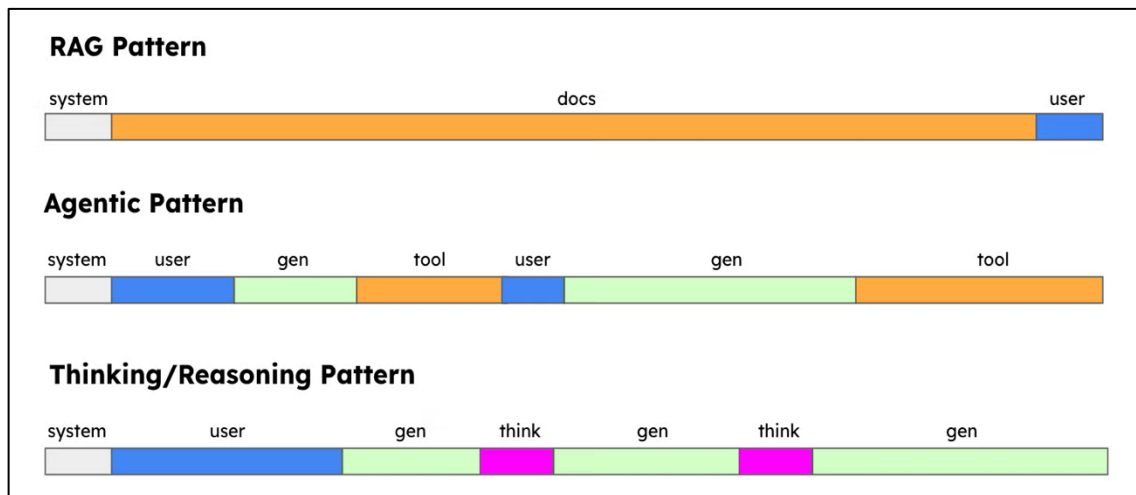
<https://pytorch.org/blog/hybrid-models-as-first-class-citizens-in-vllm/>

Remember: everything is linear algebra



ingest demo?

RAG - CONTEXT

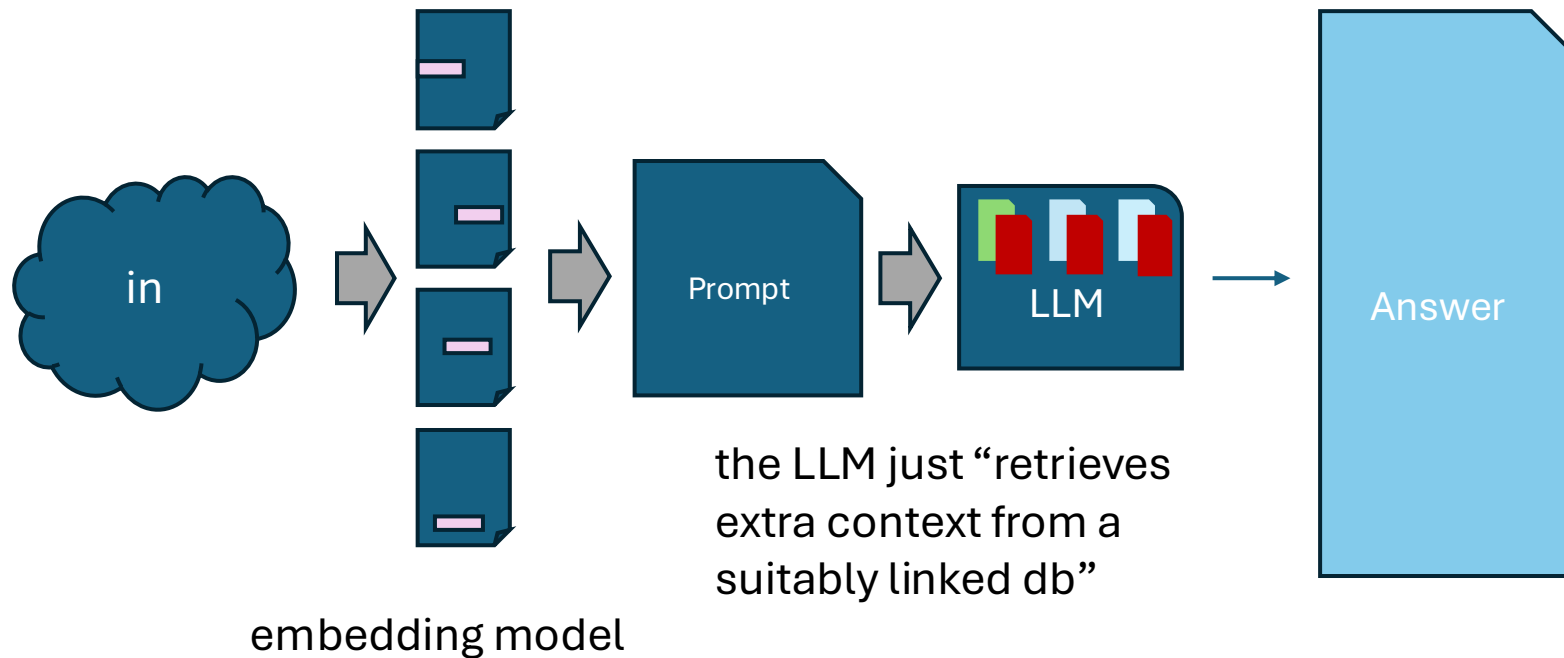


<https://pytorch.org/blog/hybrid-models-as-first-class-citizens-in-vllm/>

To find “sources”, “references” in databases, compute angle between prompt and avail docs!

(it’s literally called cosine similarity)

RAG (with a demo?)



SVD and low-rank updates

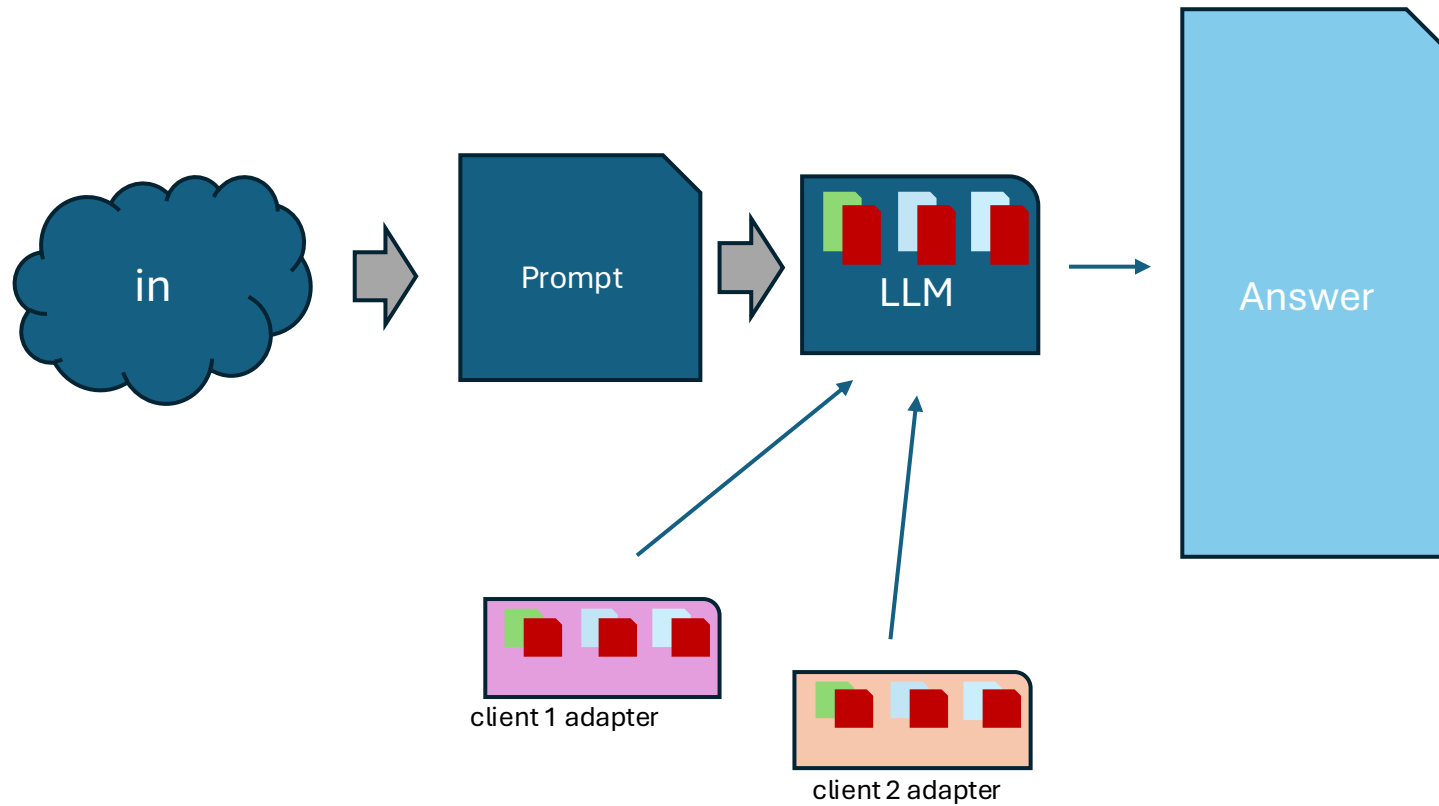
- any matrix A with dimension (m, n) can be decomposed as
- $A = U S V^T$
- U as (m, n) S as (n, n) V as (m, n)
- and we can reconstruct A without the full matrices (with some quantifiable information loss)
- we even have a short demo: take a picture
- Idea is: reconstruct vector spaces that have
- highly redundant data with by relying on “subspace”

```
20 import cv2
19 import numpy as np
18 import matplotlib.pyplot as plt
17
16 if __name__ == '__main__':
15     cam = cv2.VideoCapture(0)
14     _, frame = cam.read()
13     cam.release()
12
11     img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
10     U, S, Vt = np.linalg.svd(img, full_matrices=True)
9
8
7     k = 10
6     recon = U[:, :k] @ np.diag(S[:k]) @ Vt[:, :k, :]
5     print("matmul is", U[:, :k].shape, np.diag(S[:k]).shape, Vt[:, :k, :].shape)
4     print("=", recon.shape)
3     fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
2     ax1.imshow(img, cmap="grey"); ax1.set_title('Original')
1     ax2.imshow(recon, cmap="grey"); ax2.set_title(f'Rank-{k}')
1 ax3.imshow(np.abs(recon-img),); ax3.set_title(f'diff')
1
2 plt.show()
```

LoRA and other finetuning approaches use a similar idea! («PeFT»)

- Training is expensive (unavailable to us for “usual-scale” data)
 - -> mem required for full-scale training $\sim 2 \times \text{param} + 1 \times \text{param}$
 - (1 x fwd, 1 x bwd, 1x state ... bad approximation but it's the OOM)
- we would also need to multiply this – DDP/FSDP – and replicate for large data
- Post-training approaches of interest here: LoRA, QLoRA
- AND: We can enhance TP with them (as they're now small)
- operators vLLM lets us plug in on the fly!
- More difficult: RLHF and full-state posttraining training

Finetuning



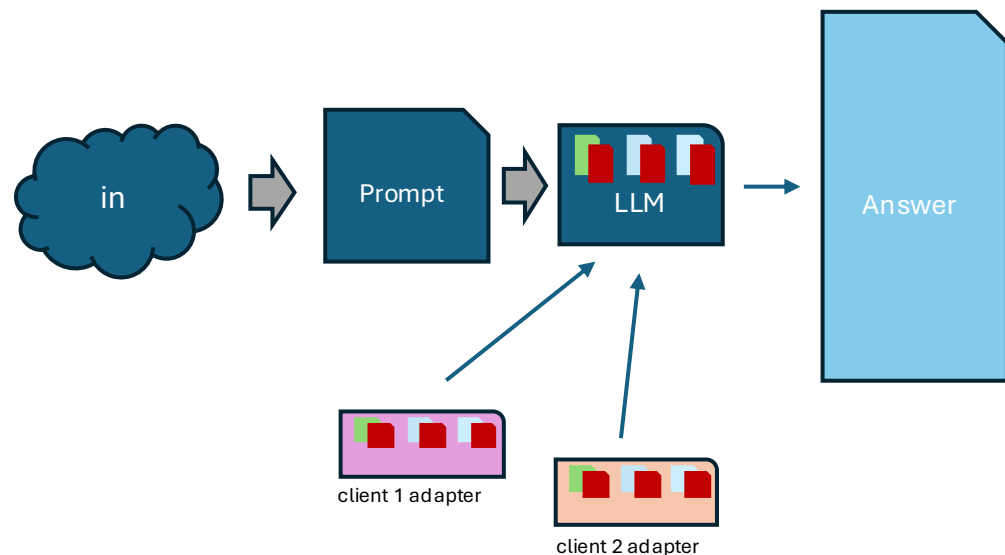
Finetuning – data and how?

```
{  
  "conversations": [  
    {  
      "from": "system",  
      "value": "You are a helpful, unbiased AI assistant."  
    },  
    {  
      "from": "gpt",  
      "value": "Hi, welcome to this interaction! I can help with your Go programming and Godoc related inquiries."  
    },  
  ],  
}
```

quick demo, easy serving?

finetuning is not trivial:

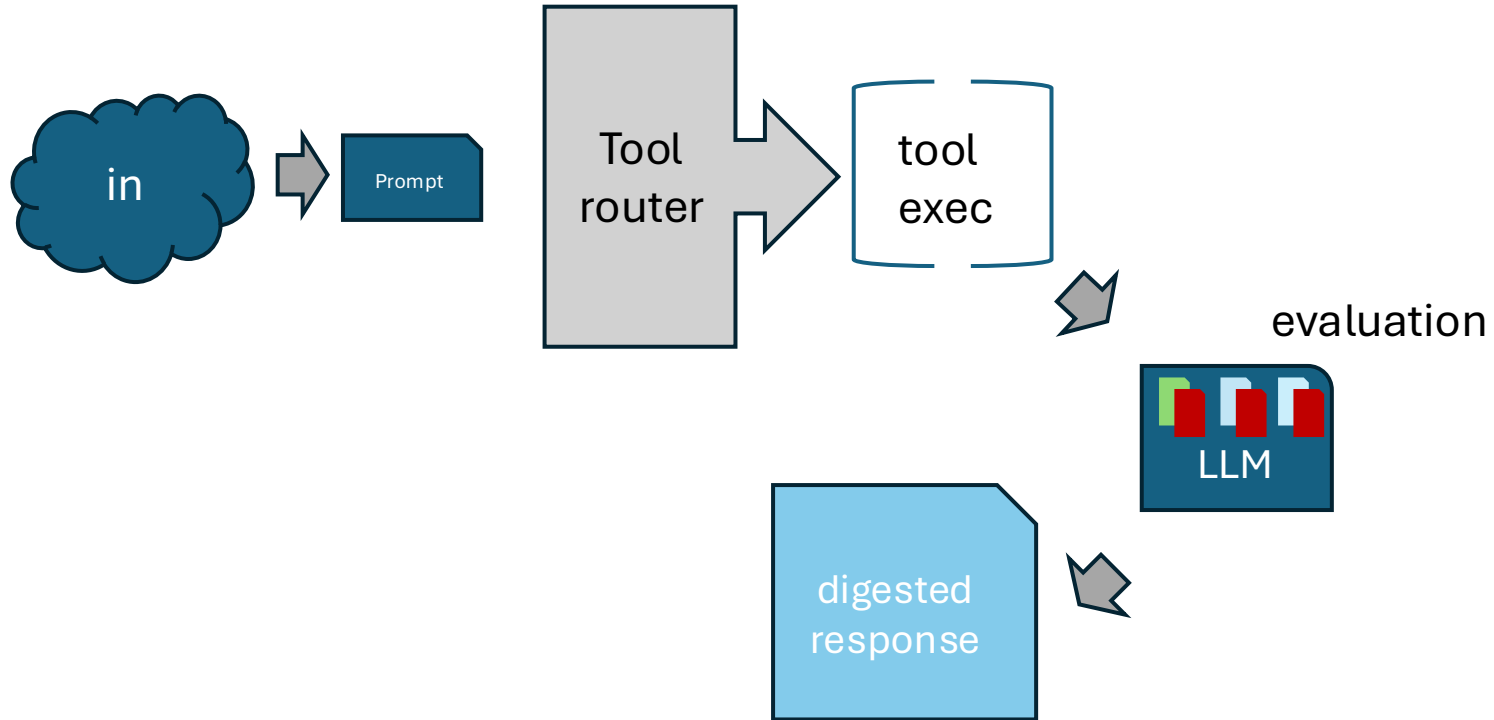
- data curation
- quantization
- model behavior in finetuning
- precision requirements?



Agents

- “tool use”, “MCP” ...?
- Lots of tools and ideas to bring “programmable AI” safely to life.
- Most are just protocol ideas how to structure the model’s interaction with
- some kind of environment.

Agents

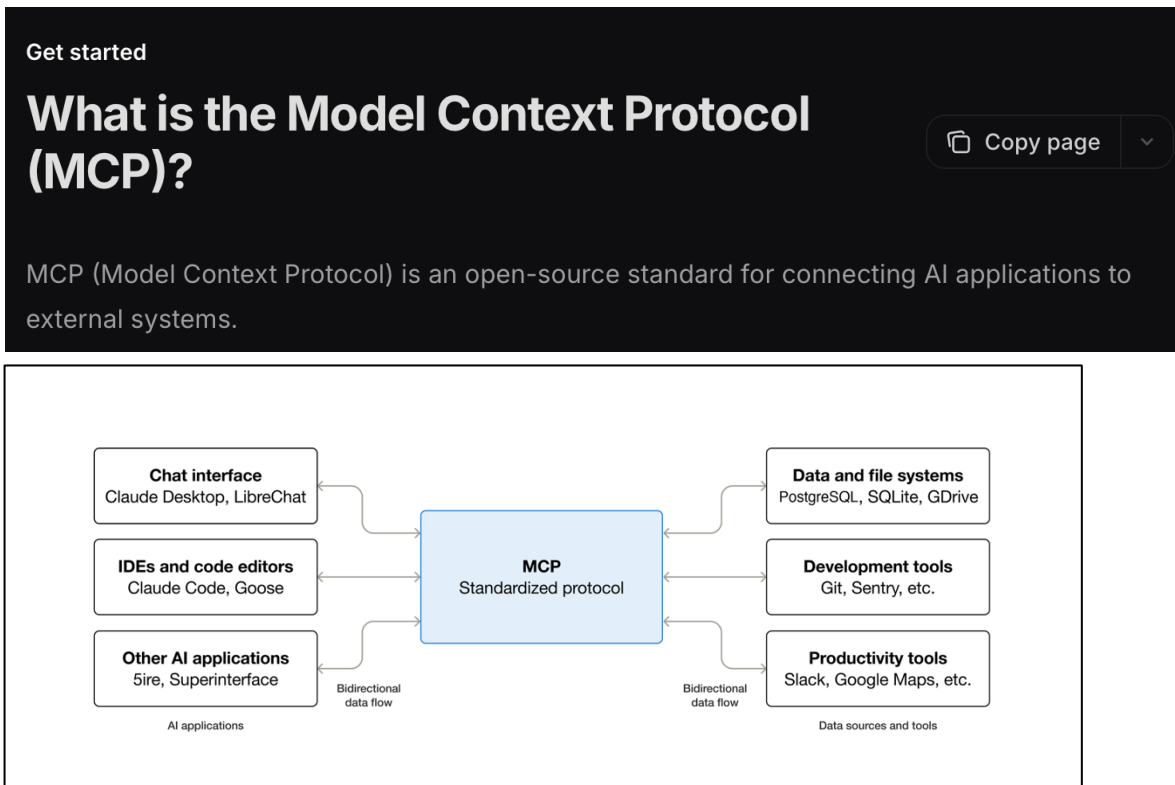


More use cases: Generative and other

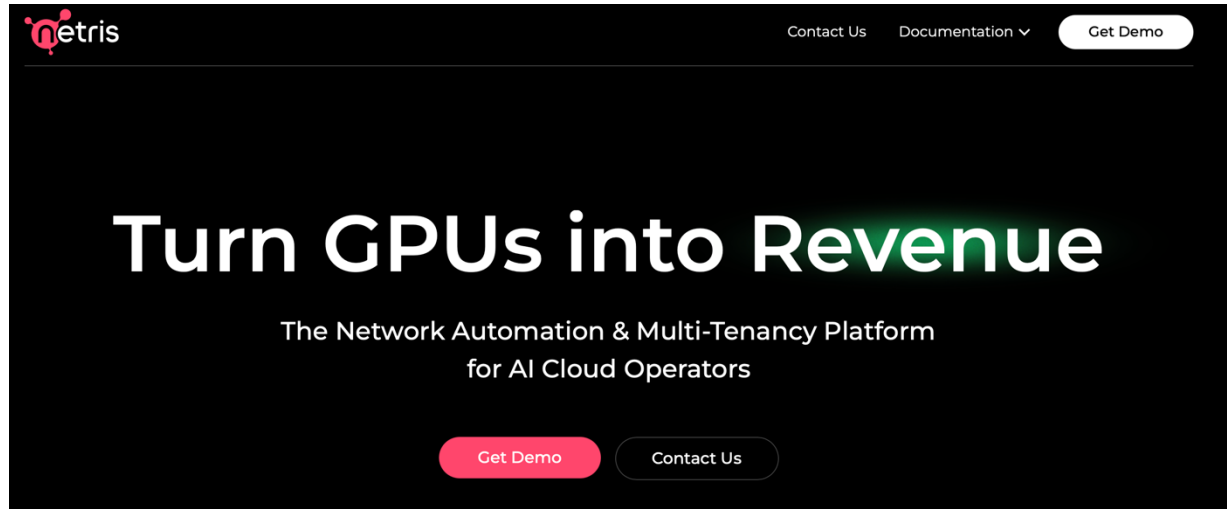
- models: base, instruct, reasoning
- distillation
- compression
- RLHF
- RL environments

Top and bottom:

<https://modelcontextprotocol.io/docs/getting-started/intro>



More use cases: Generative and other



<https://netris.io>

Takeaways

- systems engineering is where it's at
- we have a mostly solved setup for most LLM-related applications
- quantization can bite us with brittle tooling
- "agents" are just <smartly> chained prompts within controlled environments

Fragen?



Links

- <https://www.stepping-stone.ch/>
- <https://www.stoney-backup.com/>
- <https://www.stoney-cloud.com/>
- <https://www.stoney-mail.com/>
- <https://www.stoney-meet.com/>
- <https://www.stoney-office.com/>
- <https://www.stoney-services.com/>
- <https://www.stoney-storage.com/>
- <https://www.stoney-wiki.com/>

stepping stone AG

Wasserwerksgasse 7

CH-3011 Bern

Telefon: +41 31 332 53 63

www.stepping-stone.ch

info@stepping-stone.ch