

Serving Local Models with vLLM



(James) Kanin Kearpimy
Core Team – vLLM Bangkok
Fellow – Python Software Foundation

python SOFTWARE FOUNDATION

vLLM BANGKOK

MVP Microsoft® Most Valuable Professional

MongoDB Community Champion

DEMO

Should we go for local model or close model?

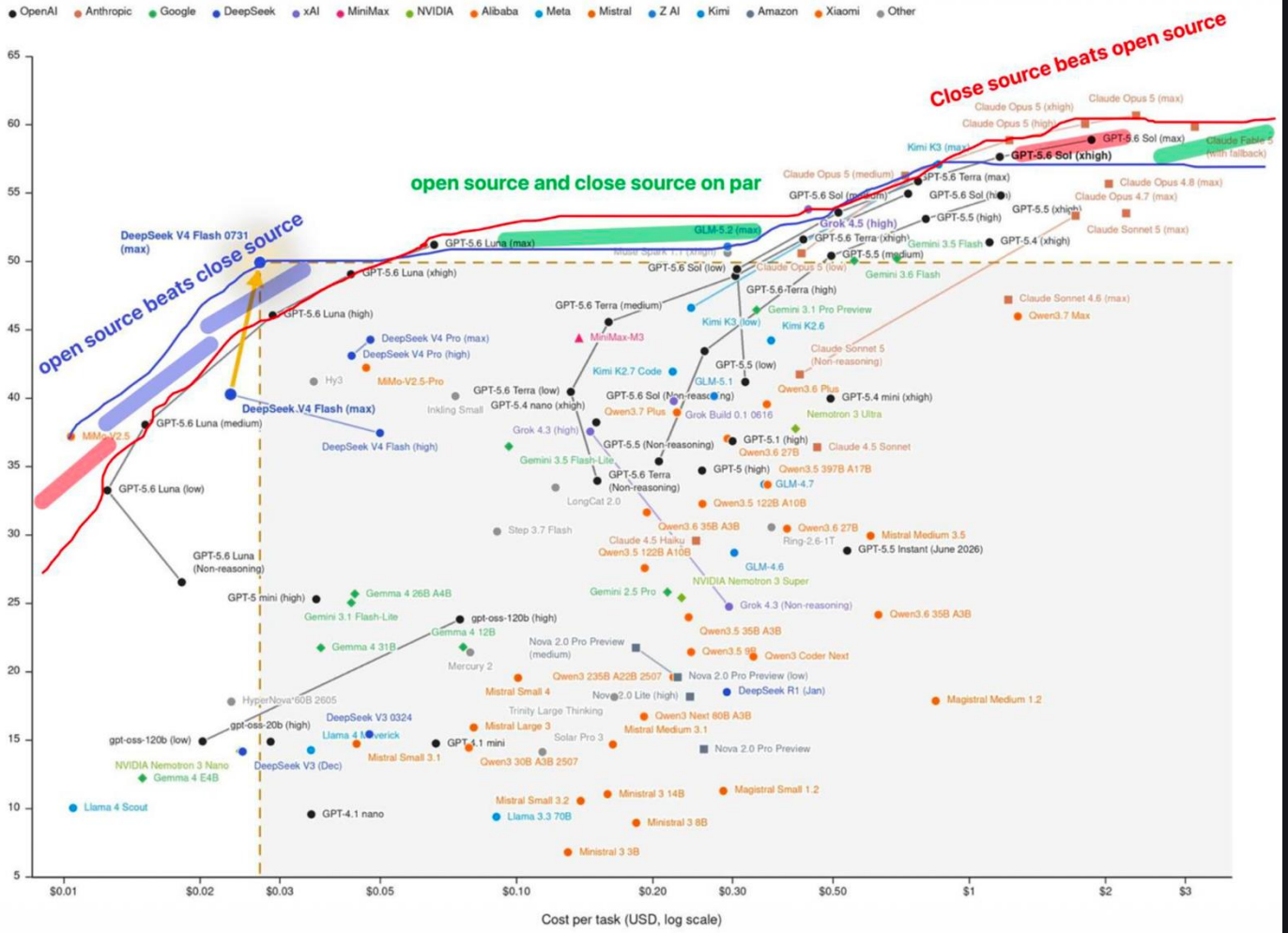
~~Should we go for local model or close model?~~

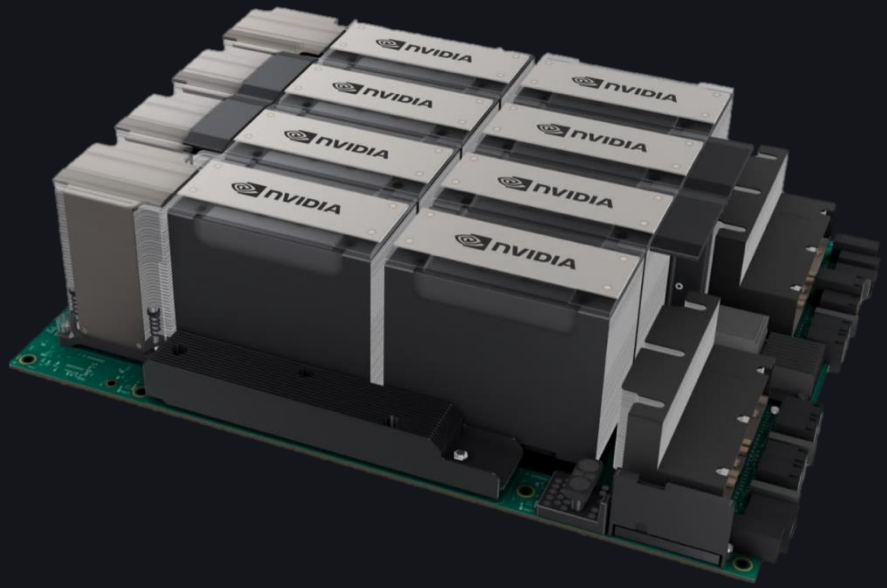
Starting point to evaluate this question.

Artificial Analysis Intelligence Index v4.1

All 128 models from the supplied CSV · weighted cost per Intelligence Index task

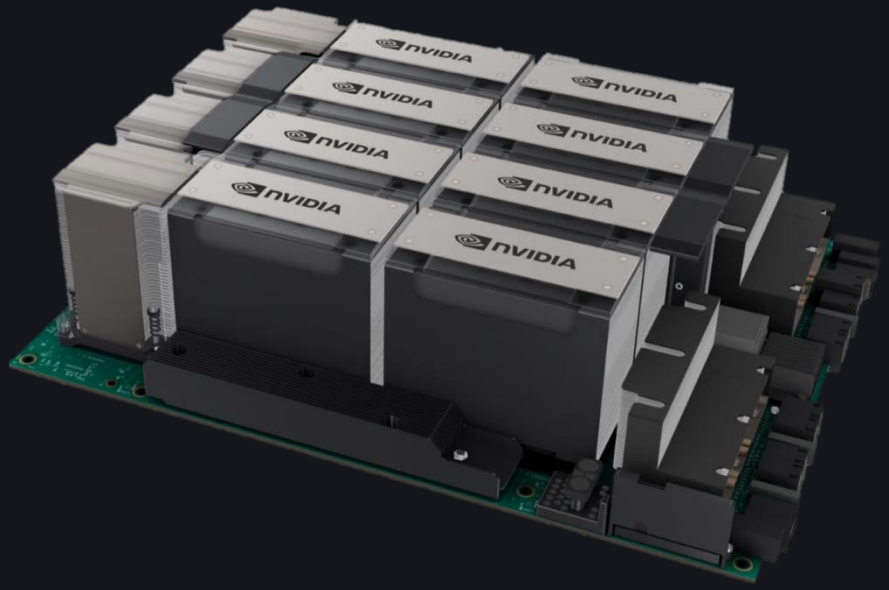
Source: Artificial Analysis · extracted 2026-07-31





~~Does GPU make us rich?~~

Buying more GPU resolve everything?



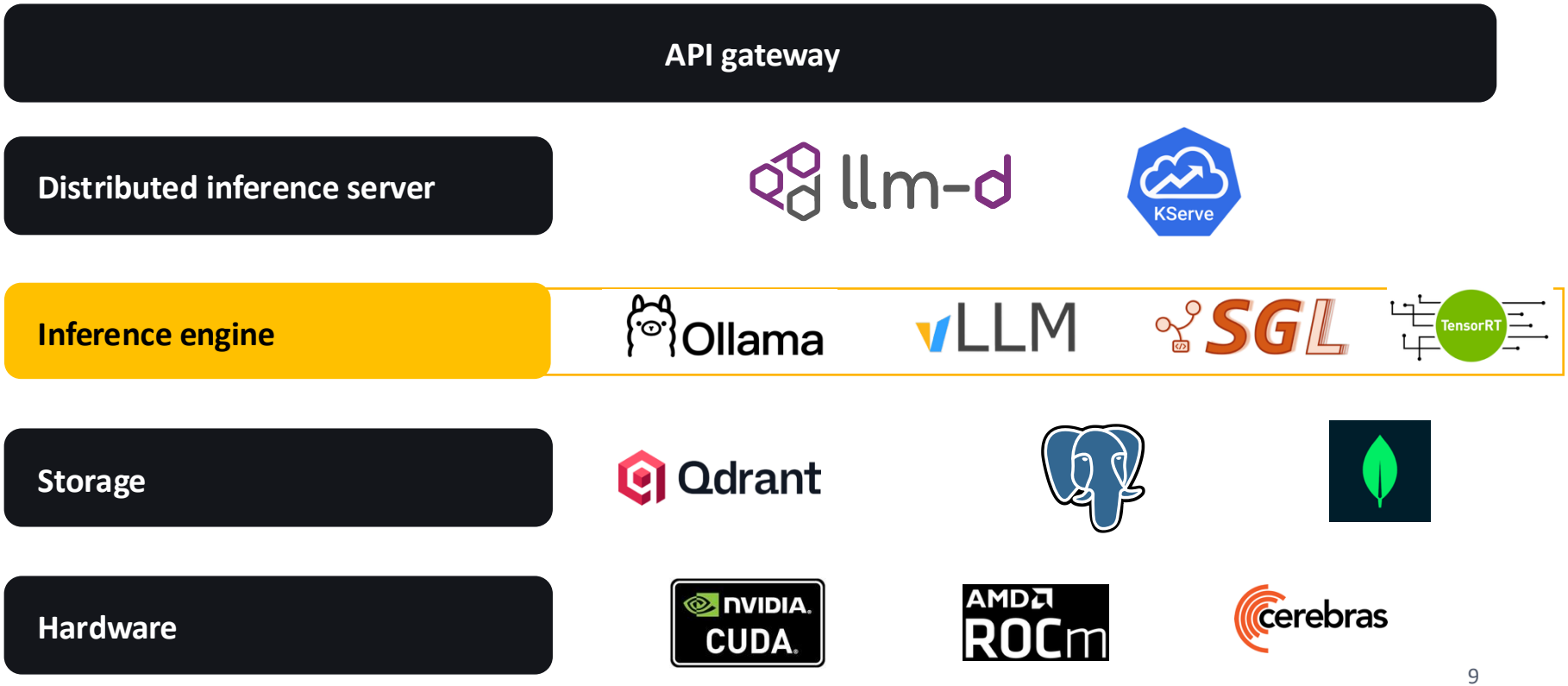
~~Does GPU make us rich?~~

~~Buying more GPU resolve everything?~~

Partially true and we need more than hardware

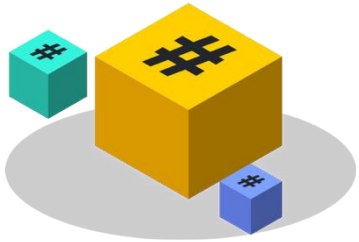
What happens behind the *engineering* scene?

AI (LLM) Stack


API gateway
Distributed inference server
llm-d
KServe
Inference engine
Ollama
vLLM
SGL
TensorRT
Storage
Qdrant
Hardware
NVIDIA
CUDA
AMD
ROCm
cerebras

Have you ever experienced these situation?

why and how?



Do I really need three H100 GPU?

My GPU is 100% utilization, but my LLM service has less throughput?

I have the same system prompt and my LLM always take long processing time for every requests?

Choosing a serving engine

Ollama serve

Simple local setup

SGLang

Fast serving runtime

TensorRT

NVIDIA-optimized

vLLM

Our pick for this talk

Why vLLM?

Active community

90,000

GitHub stars

Industry-grade models

Llama · Qwen · DeepSeek · Mistral · Gemma

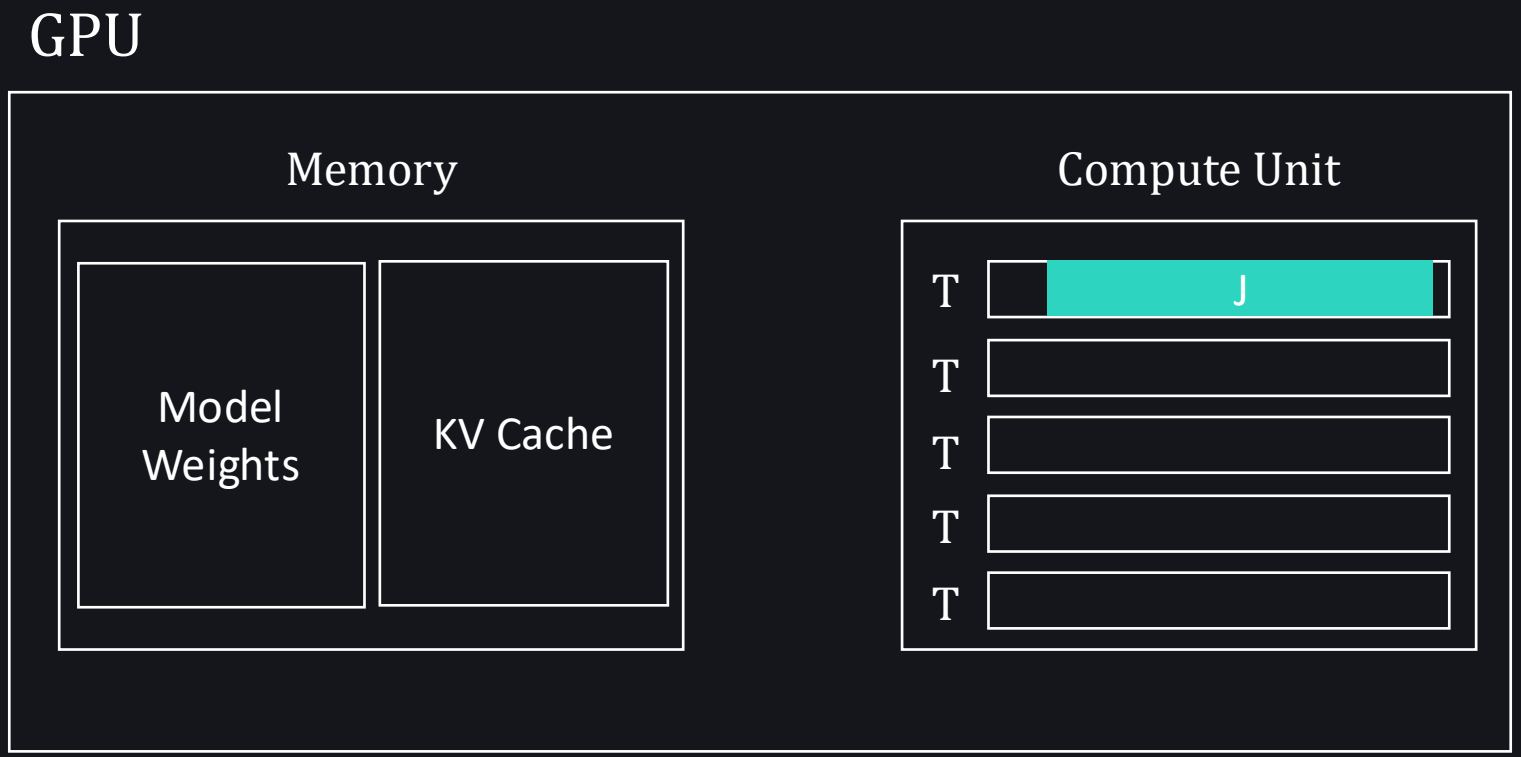
Major open model families

Hardware range

GPU · CPU · TPU

Industry-grade accelerators

Background


GPU
Memory
Model Weights
KV Cache
Compute Unit
T
J
T
T
T
T
T = Thread
J = Job

Four ideas inside vLLM



Continuous batching

Keep the GPU fed



PagedAttention

KV cache in small blocks



Prefix caching

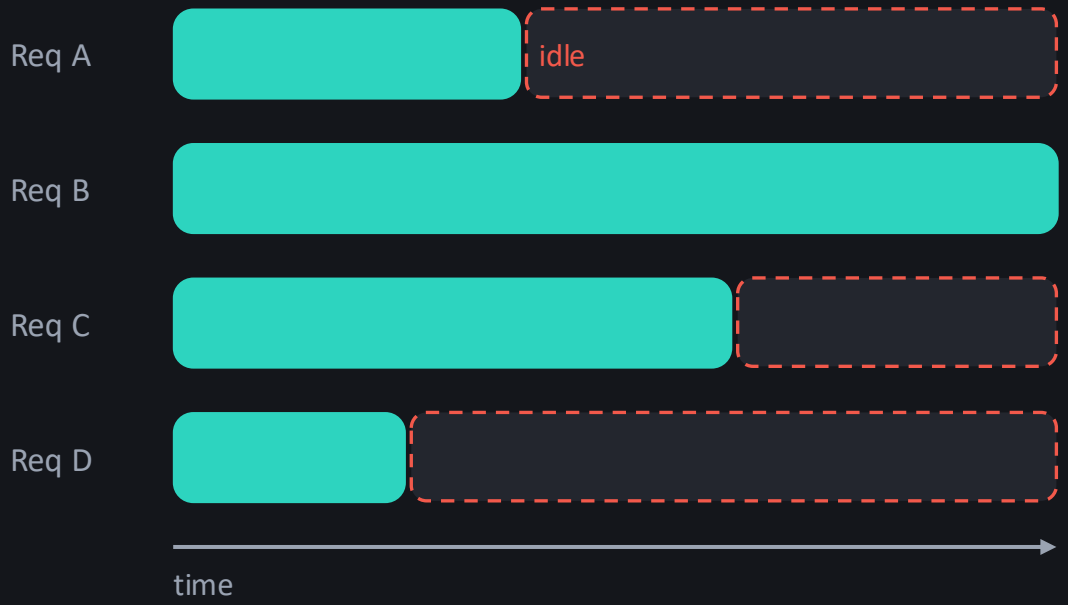
Reuse repeated prompts



Quantization

Smaller weights, faster answers

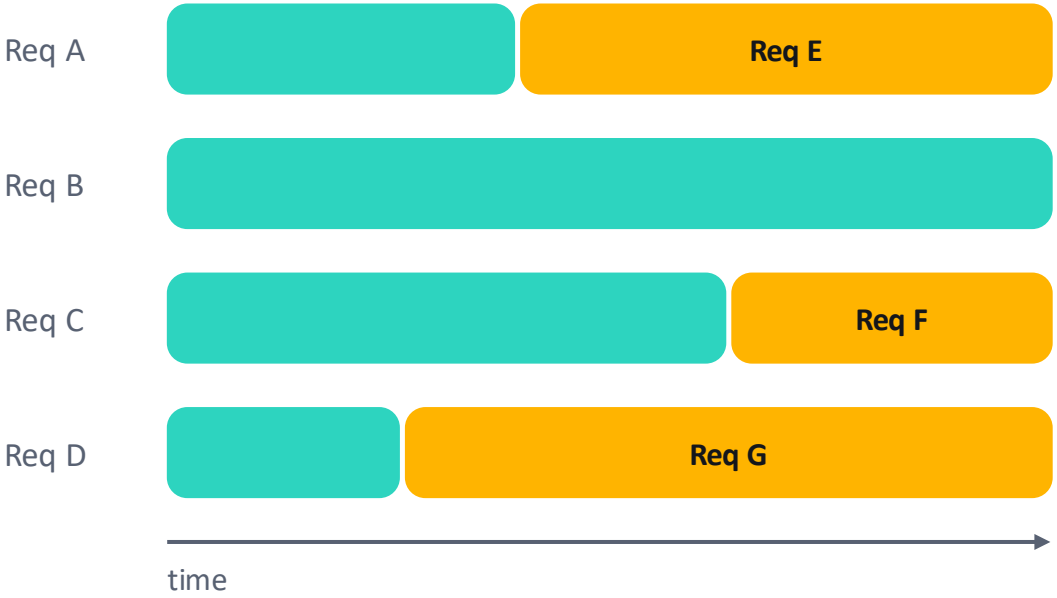
The GPU sits idle



Wait for the slowest

A batch ends only when its longest request ends. Short requests finish early and their slots stall while the GPU still has capacity.

Continuous batching



Refill the slot

As soon as a request finishes, a waiting one takes its place. Work is fed to the GPU continuously to maximize utilization.

Four ideas inside vLLM



Continuous batching

Keep the GPU fed



PagedAttention

KV cache in small blocks



Prefix caching

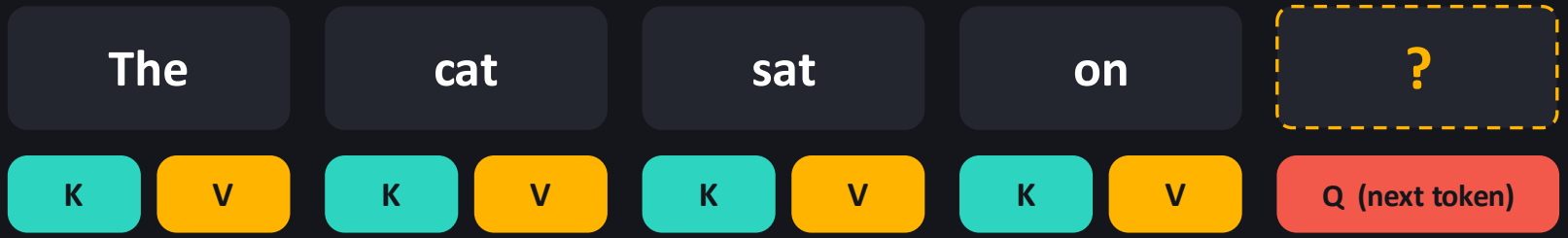
Reuse repeated prompts



Quantization

Smaller weights, faster answers

Attention and the KV cache



Each new token's query reads the keys and values of every earlier token.

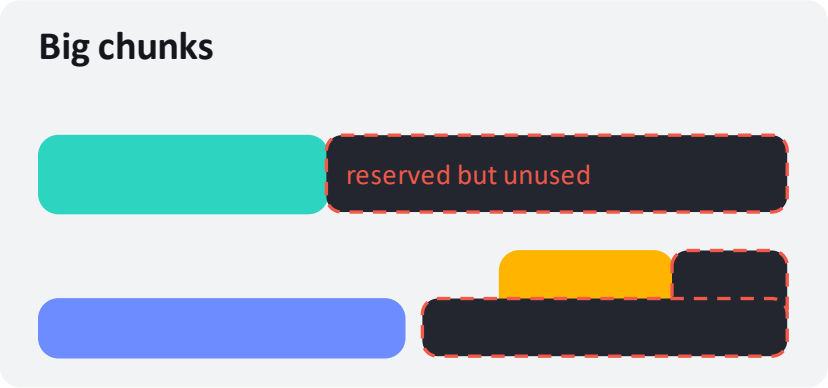
No cache

Recompute K and V at every step. Wasteful.

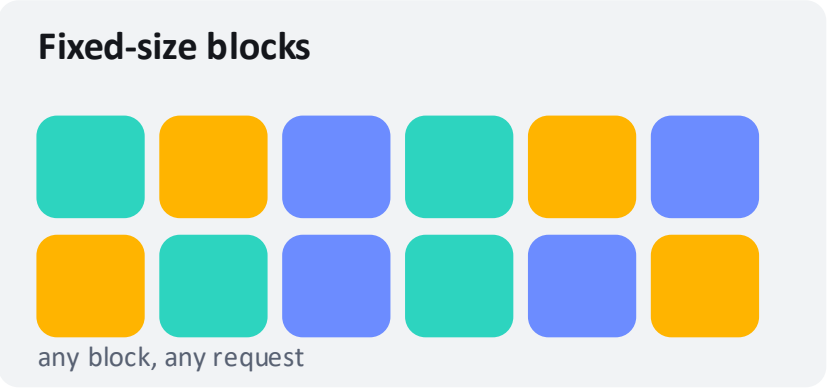
KV cache

Store K and V once, fetch later. But one huge memory chunk per request causes fragmentation.

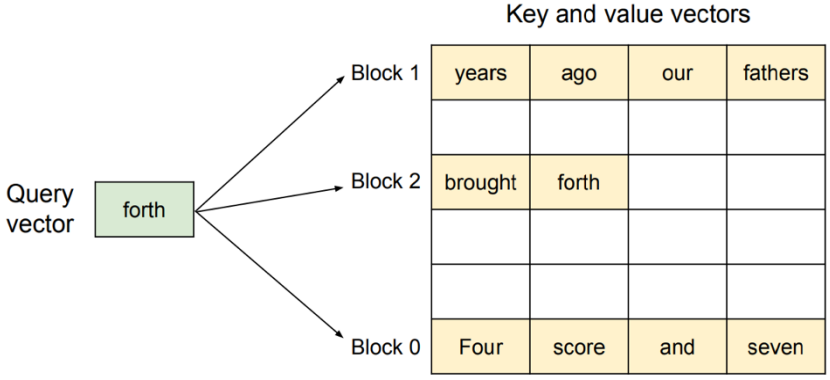
Contiguous Chunks

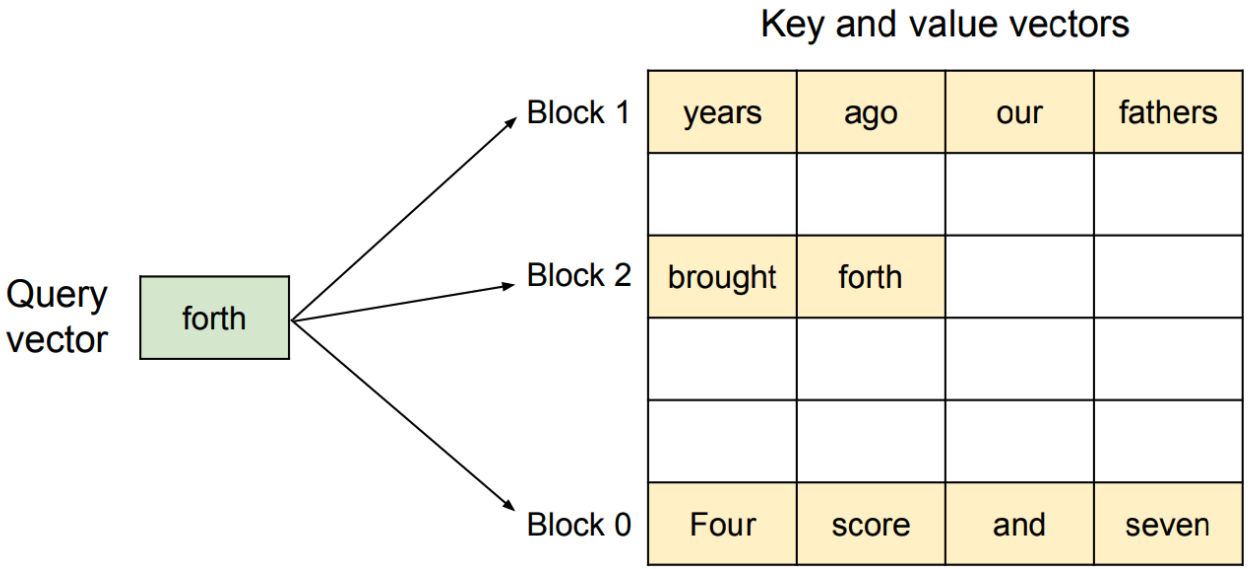


PagedAttention



- Pre-allocate fixed-size blocks, like virtual-memory pages in an OS
- A cache manager stores and fetches the KV blocks
- Small blocks mean minimal memory fragmentation





- Pre-allocate fixed-size blocks, like virtual-memory pages in an OS
- A cache manager stores and fetches the KV blocks
- Small blocks mean minimal memory fragmentation

Four ideas inside vLLM



Continuous batching

Keep the GPU fed



PagedAttention

KV cache in small blocks



Prefix caching

Reuse repeated prompts



Quantization

Smaller weights, faster answers

Same text, no reuse



Attention handles request by request. Even when two requests carry identical text, the computed KV is not shared.

Prefix caching



Hash each KV block. Match the new request's tokens to existing blocks and reuse them: less computing, faster answers.

reused computed

Four ideas inside vLLM



Continuous batching

Keep the GPU fed



PagedAttention

KV cache in small blocks



Prefix caching

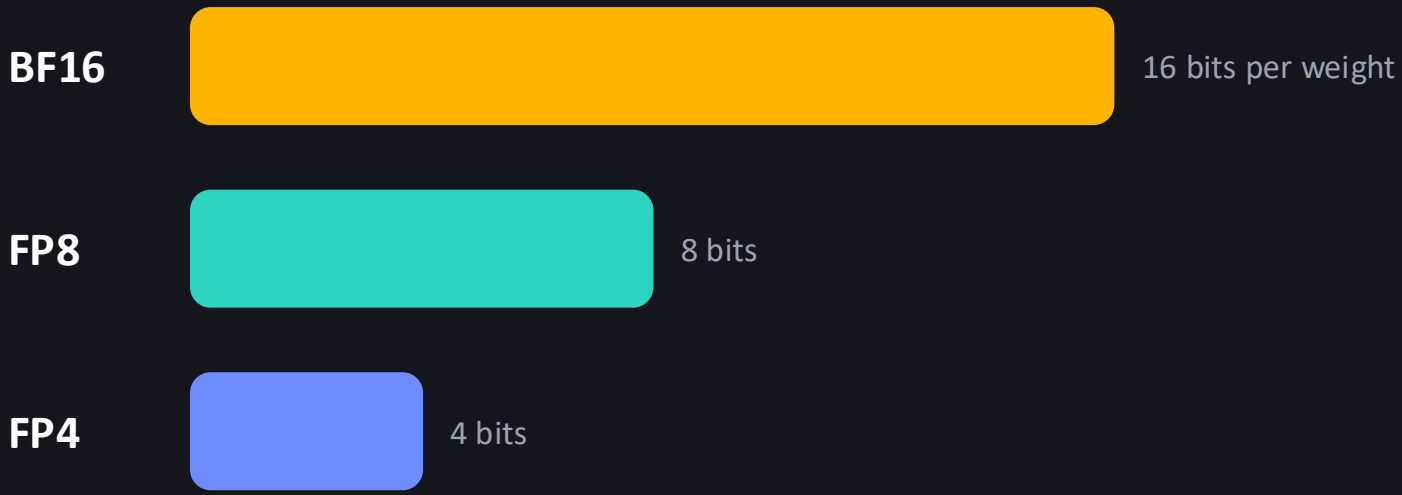
Reuse repeated prompts



Quantization

Smaller weights, faster answers

Model size and weights



Weights predict the next token

Compress them to shrink the model, trading a little answer quality for size.

Quantize by task

Generic chat

Quick response beats a smarter answer.

Quantize

FP8 **FP4**

Long-running agents

Sensitive workflows need accuracy.

No quantization, or carefully chosen

BF16

Request: “Please introduce yourself”.

Inference Engine (vLLM)

Prefix Cachine

Qwen2.5

PagedAttention

Quantization

Continusous Batching

Memory

Model Weights

KV Cache

Compute Unit

J

T

T

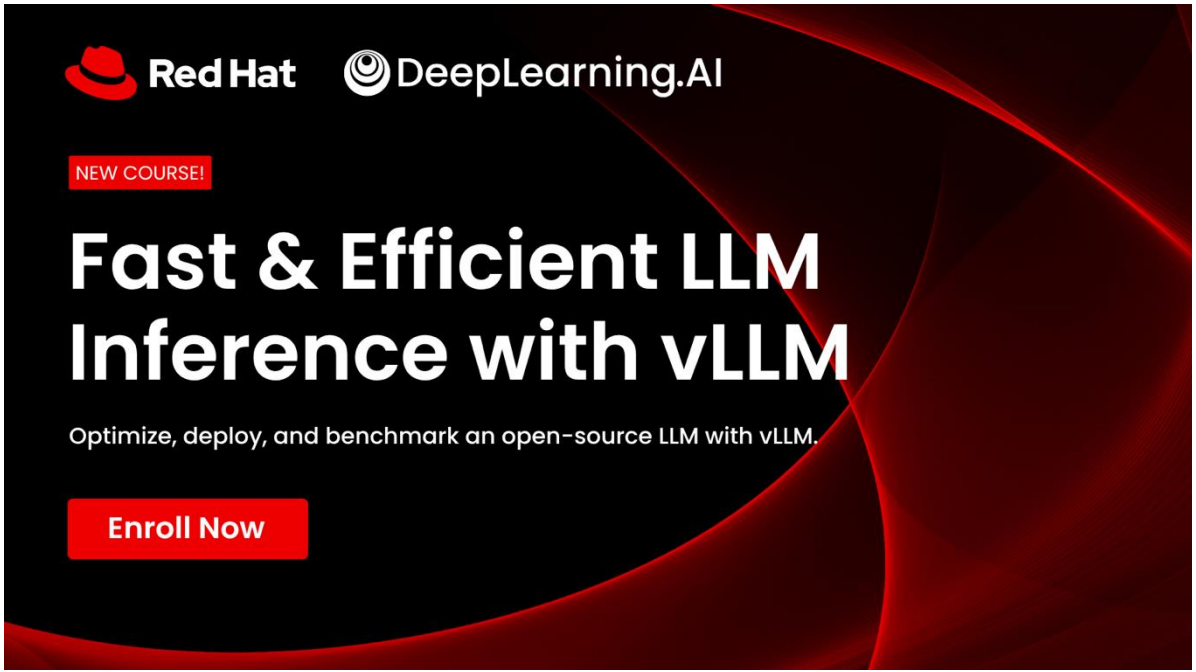
T

T

T



Next Step



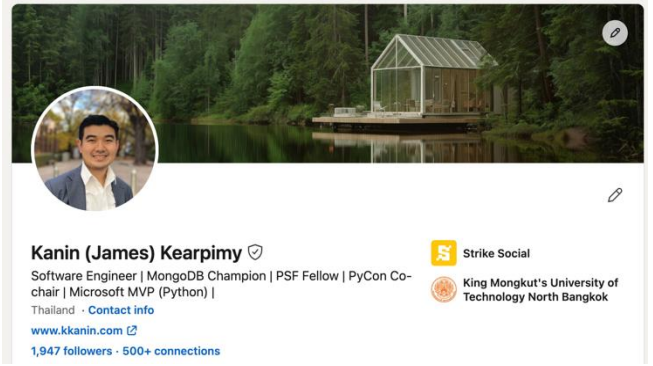
vLLM Course
(deeplearning.ai)

Next Step

Receive **Slide**
here



3 Dec 2026
Event
Code: JAMES111



Let Connect!
LinkedIn: Kanin Kearpimy