

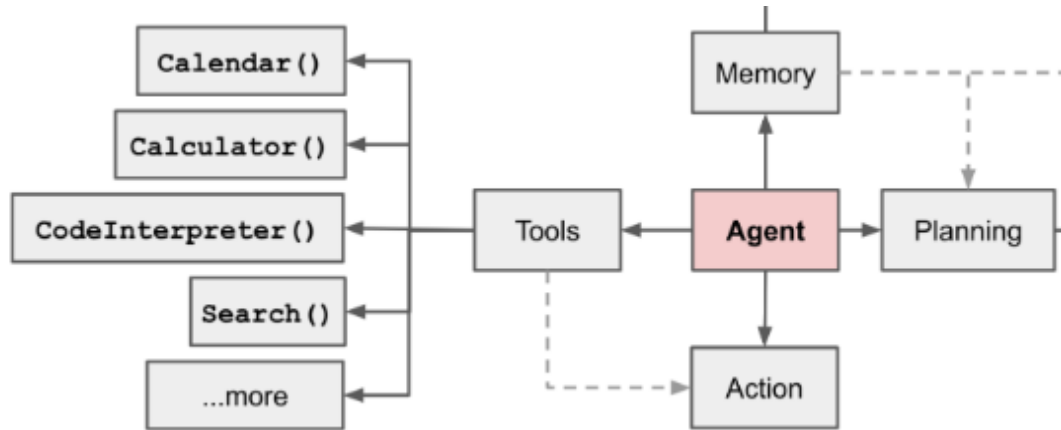
Understanding LLM Performance

Yiying Zhang

What is LLM Agents

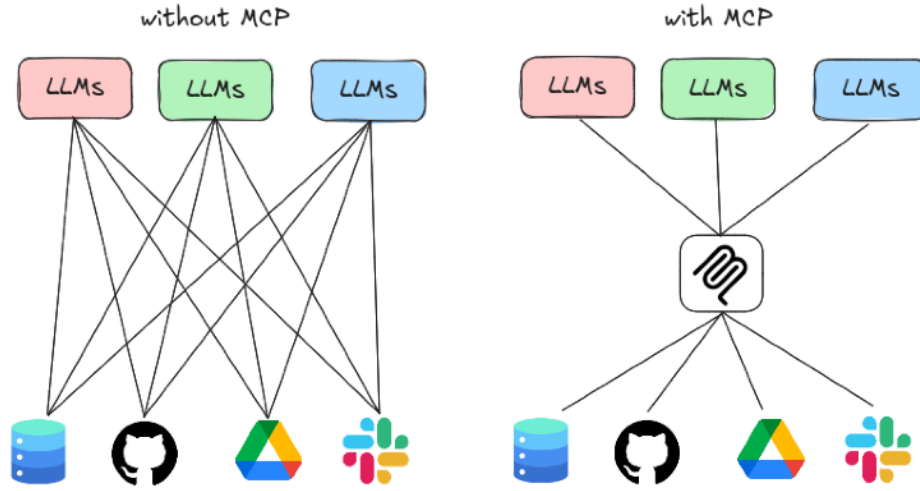
Tool use:

- The agent learns to call **external APIs** for extra information that is missing from the model weights (often hard to change after pre-training), including current information, code execution capability, access to proprietary information sources and more.



MCP (Model Context Protocol)

- Connecting (N) LLMs to (M) external tools/resources used to be a NxM problem
- MCP standardizes the LLM-tool communication into a N->1->M process
- Build with a client-server model
 - MCP client: the agent that needs to call tool/data
 - MCP server: a service to expose external tools and data sources



AI Agent/Workflow Frameworks

- Frameworks initially proposed to standardize AI workflows, provide some out-of-box design patterns and abstractions
- Some examples
 - LangChain: Came out the earliest, probably the most popular and hardest to use
 - LlamaIndex: Good RAG support
 - CrewAI and Camel: multi-agent framework for more complex tasks
- But a lot of unnecessary, added complexity for agents, harder to customize
- My experience of what's the easiest and sufficient for many tasks
 - No framework (pure Python)
 - No MCP (can just write your own functions or hooks)
 - No A2A (no need for multi-agent)

What is AI Agent Infra?

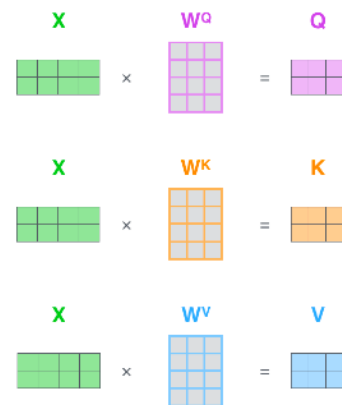
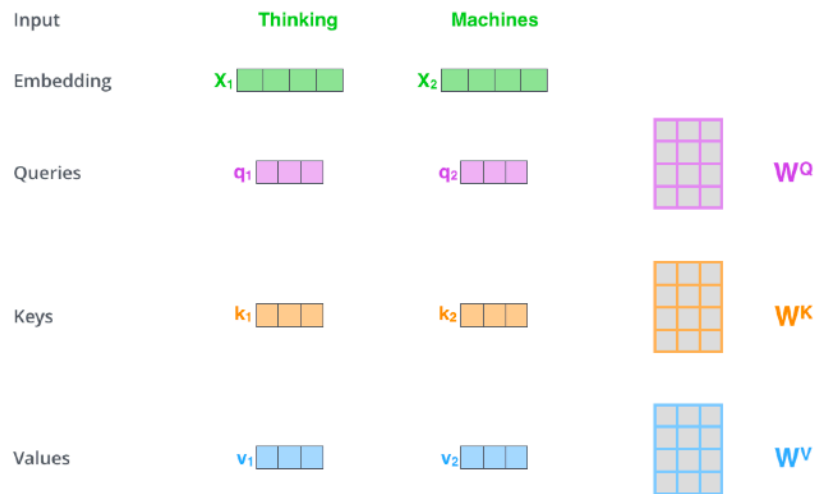
- Agent testing and evaluation
 - Unit + e2e test, metrics, benchmarks, human-in-the-loop
- Agent autotuning and optimization
 - Automated prompt tuning, model selection, tool selection, workflow optimization
- Agent hosting
 - Serverless or long-running?
 - Stateful or stateless?
- Tooling, memory, data

**Demo Time: Eigent Computer-Use
Agent performing a Discord
summarization task**

Outline

- Transformer primer
 - Introduction oriented for LLM infra (perf problems), not the theory
- LLM performance

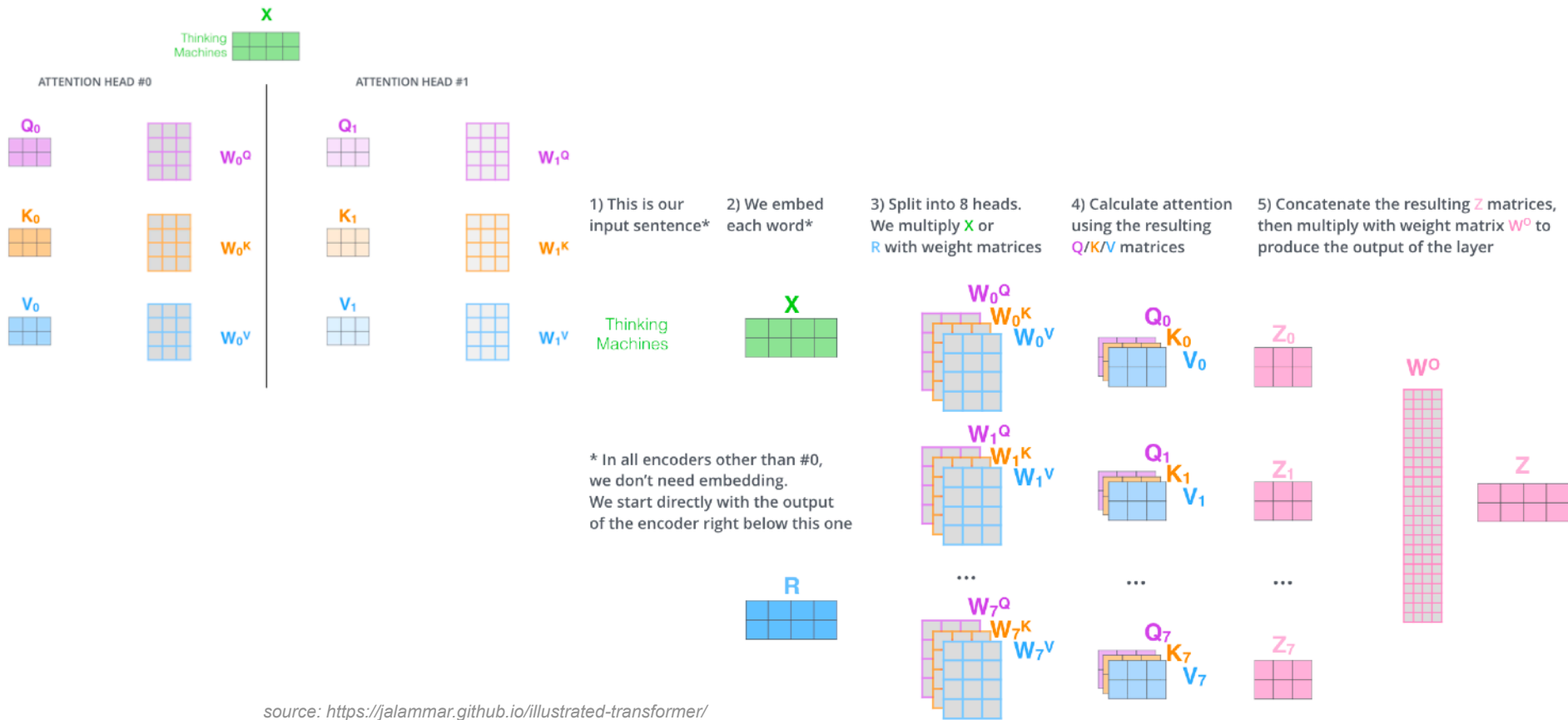
Self Attention



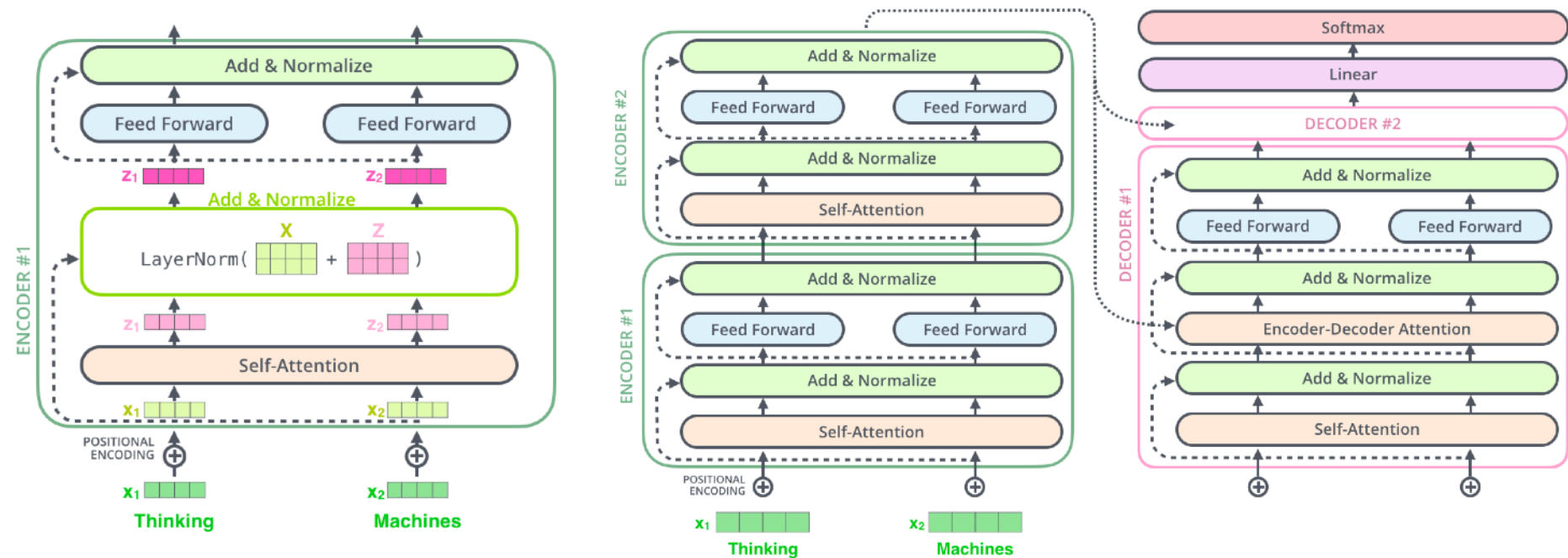
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V$$
$$= Z$$

The diagram shows the final step of the self-attention mechanism. It illustrates the multiplication of the query matrix Q (2x4) and the key matrix K^T (4x2) to produce a 2x2 matrix. This result is then divided by the square root of the key dimension $\sqrt{d_k}$. The result is passed through a softmax function, and the output is multiplied by the value matrix V (2x4) to produce the final output Z (2x4).

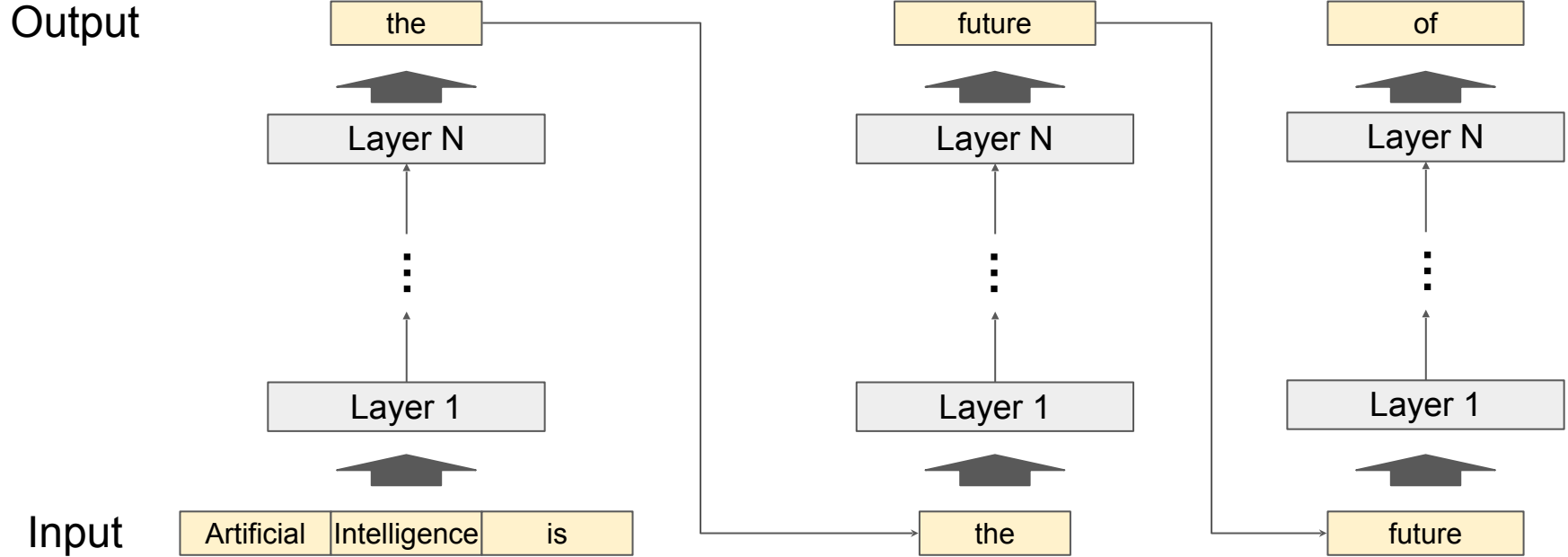
Multi-headed Attention



Transformer Model



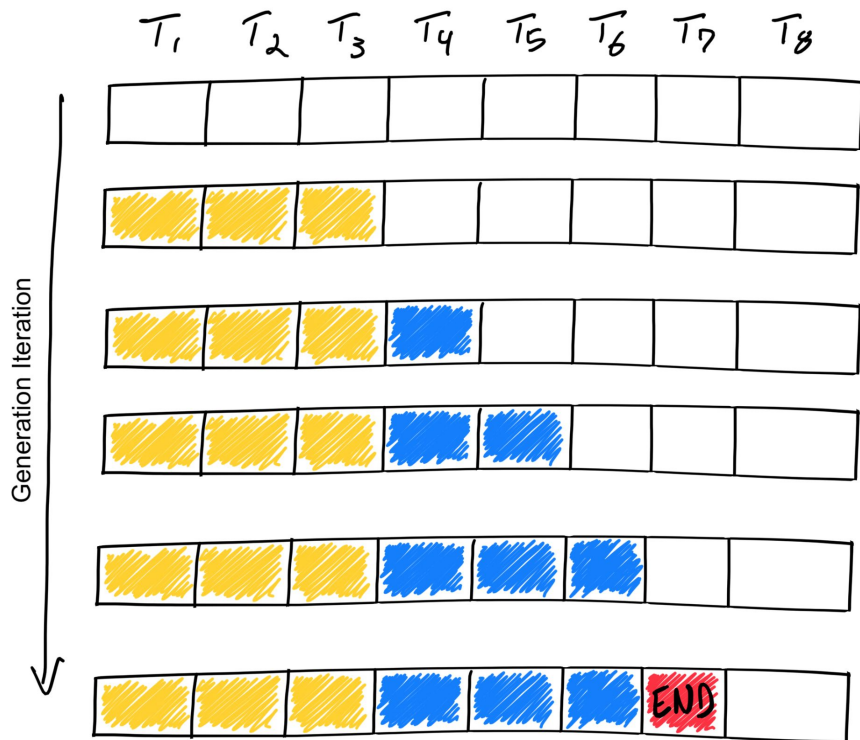
Inference process of LLMs



Repeat until the sequence

- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|end of sequence|>”)

LLM inference background



How does text generation work?

Iterative: each forward pass generates a single token

Autoregressive: generation consumes prompt tokens + previously generated tokens

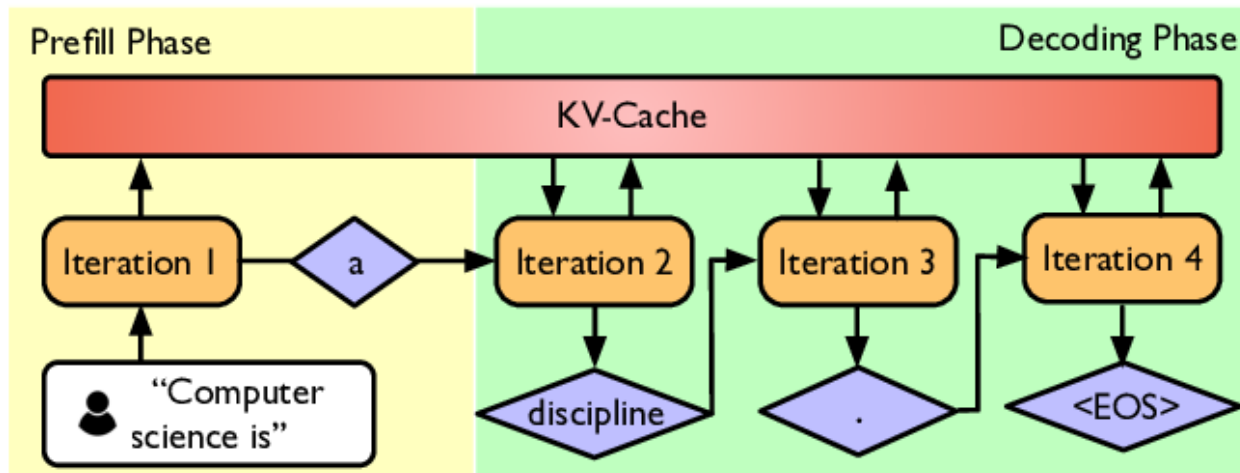
Completion potentially decided by model: A generated token can be the end-of-sequence token

Legend:

- Yellow: prompt token
- Blue: generated token
- Red: end-of-sequence token

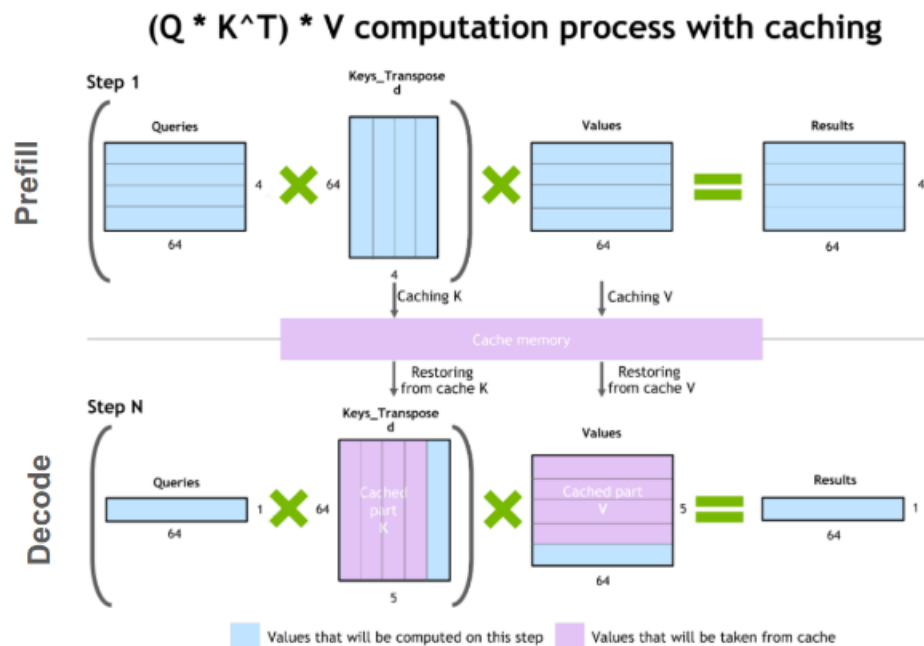
Prefill and Decoding Stages

- Prefill: processing model input (all in one forward pass)
- Decode: generating output token, one at a time



KV Cache

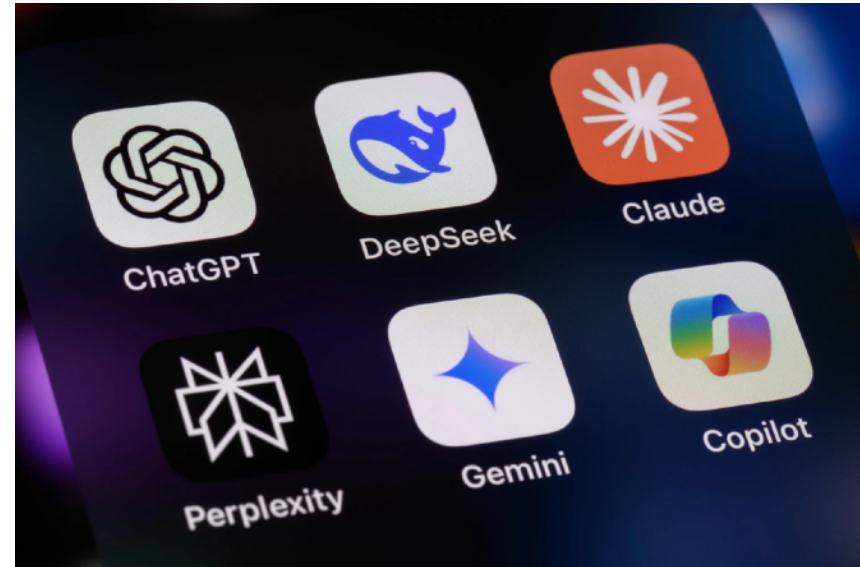
- KV Cache: stores **Key (K)** and **Value (V)** tensors computed at each transformer layer during inference to avoid recompilation
- Prefill: store computed KVs of input sequence in KV cache
- Each iteration in decode phase: each new token only needs to attend to cached KV states + the latest token

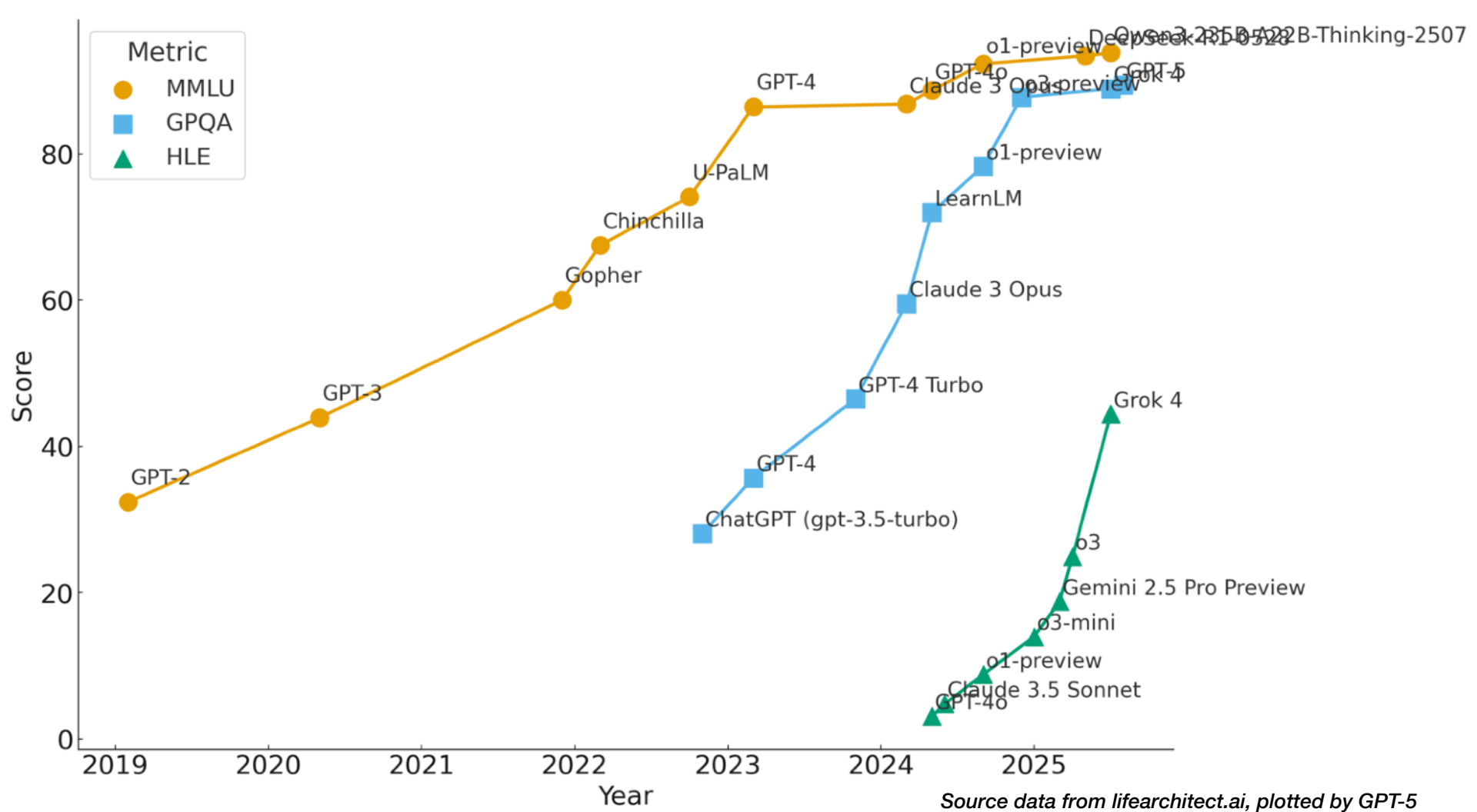


Question: what makes KV cache big?

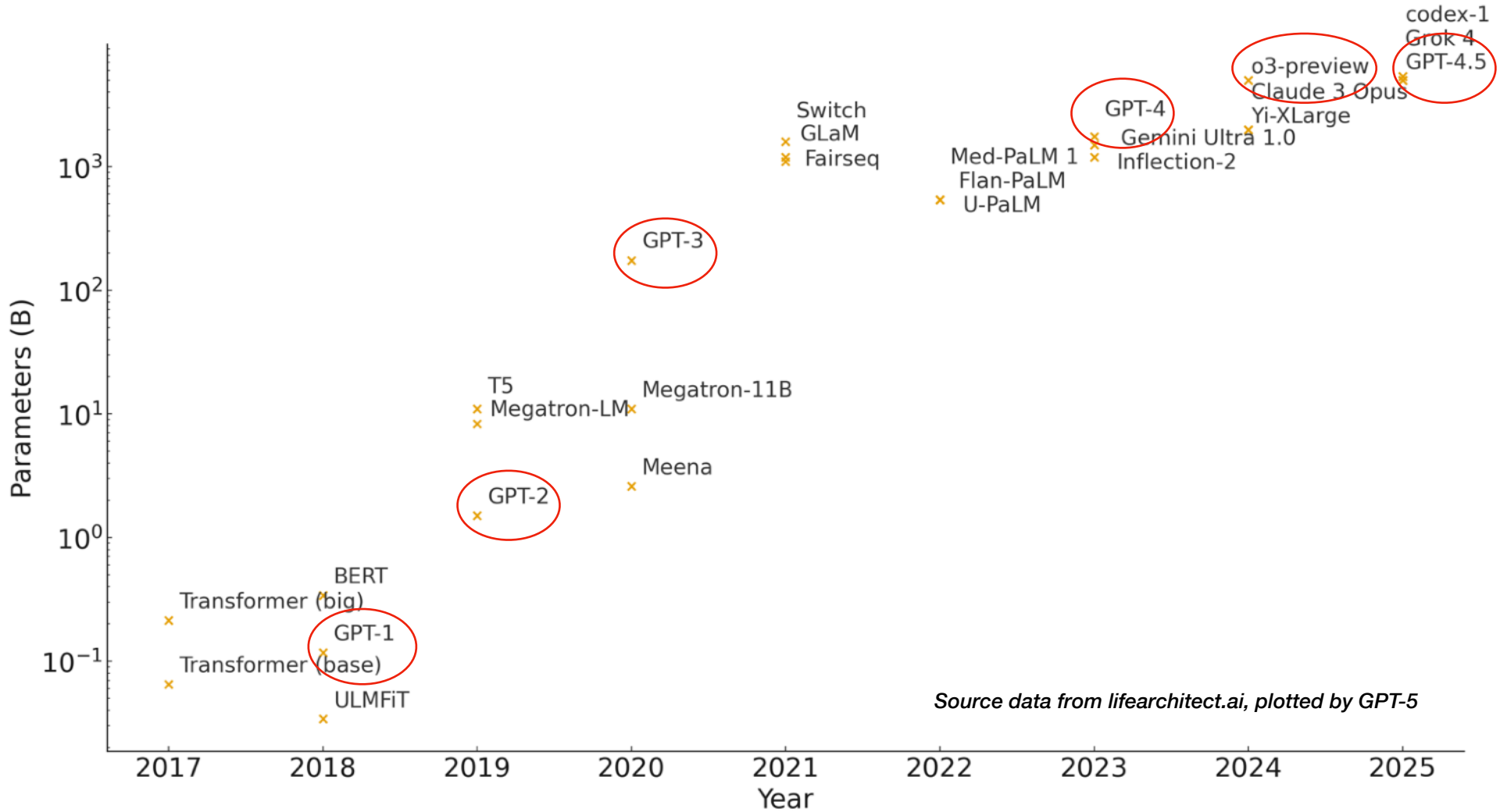
[recap] What is **LLM** (Large Language Model)?

- Language models
 - Text to text generation (i.e., text as input and text as output)
- that are large
 - Massive number of model parameters (weights)
 - Trained on huge amounts of data

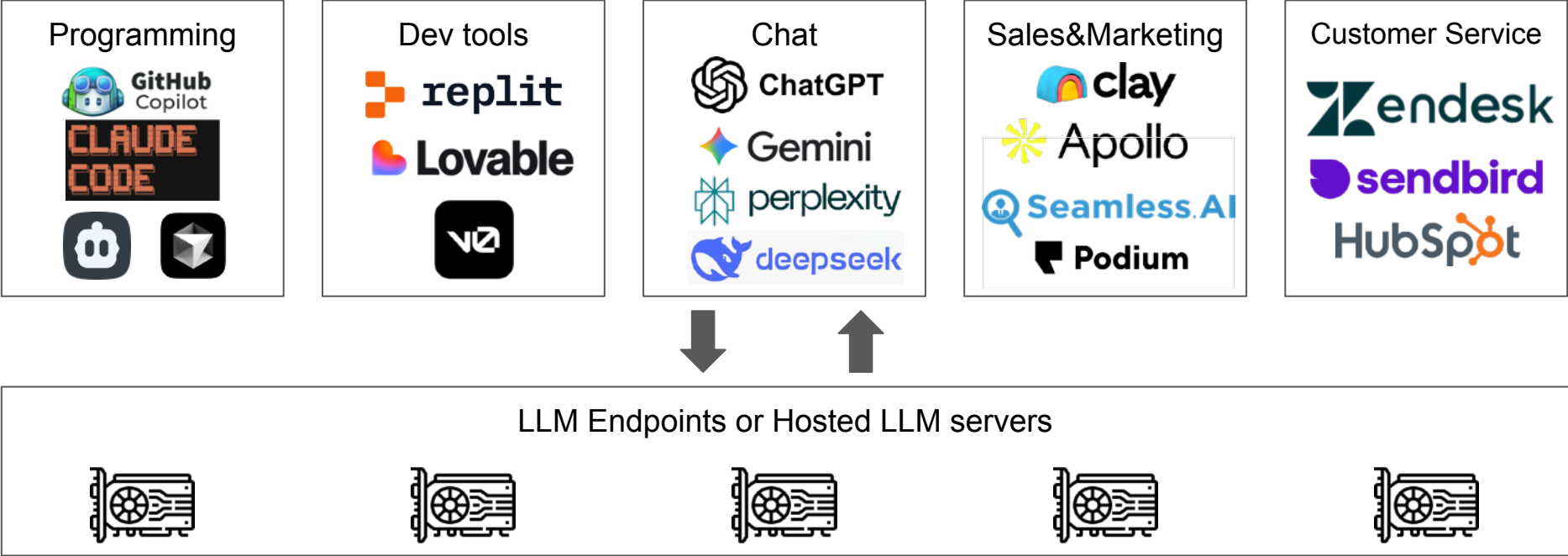




Source data from lifearchitect.ai, plotted by GPT-5



Popular LLM-powered services

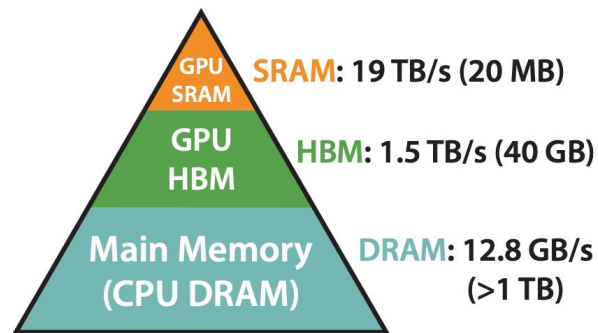


What is LLM Infra?

- LLM training
 - Pre-training (foundational model training)
 - Post-training (fine-tuning, RLHF)
- LLM serving
 - Single-GPU/CPU LLM inference
 - Distributed model serving
- LLMOps
 - Training data collection, preparation, and synthesize
 - Experimental tracking, model registry
 - Monitoring and logging of LLM serving

Systems challenges that increase cost

- Size of LLM parameters
 - Llama2 70B ~ 130GB to store float16 parameters
 - 2x A100-80GB to store, 4x+ A100-80GB to maximize throughput
- Memory IO huge factor in latency
 - For a single token, have to load 130 GB to compute cores
 - CPU memory IO ~= 10-50 GB/s
 - GPU memory IO ~= 2000 GB/s (A100 80GB)
- High throughput requires many FLOPS
 - CPU can do real-time generation of a single sequence
 - GPU can do real-time generation for many sequences



**Memory Hierarchy with
Bandwidth & Memory Size**

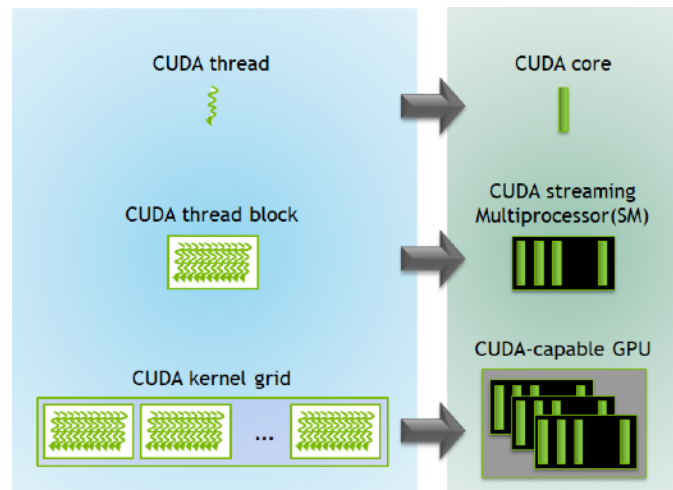
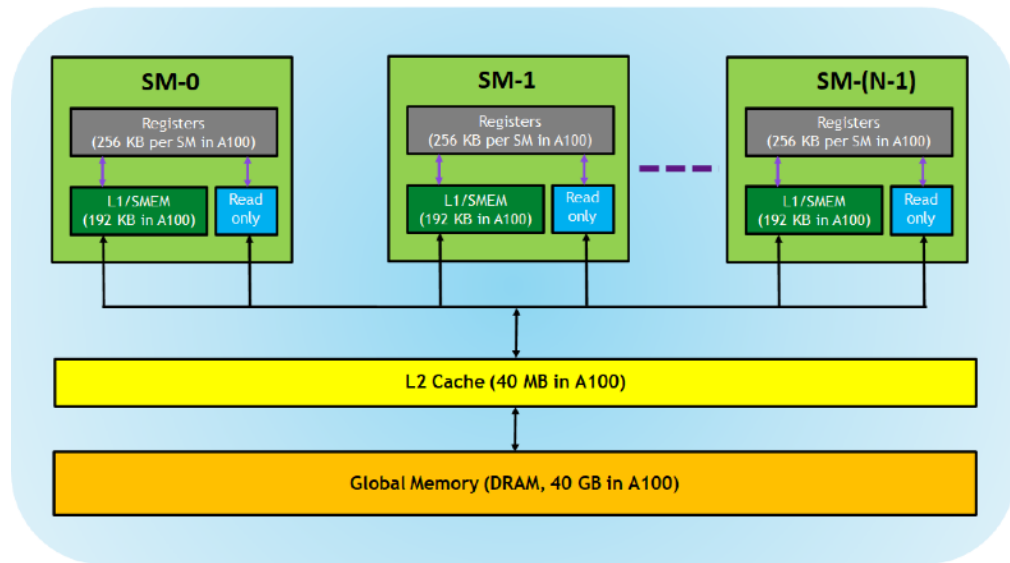
What does “Performance” mean in LLM serving?

- Not quality “performance”: in this class and in systems context in general, performance means temporal performance, not quality or accuracy
- **Time To First Token (TTFT)**: Queueing time + refill time + one token decoding: How quickly users start seeing a response, i.e., *response time*
- **Time Per Output Token (TPOT)**: Time to generate each additional output token. Average TPOT x output token length is the time users see all the response after seeing the first token
- **Latency**: The overall *request time* it takes for the model to generate the full response for a user. $\text{latency} = (\text{TTFT}) + (\text{TPOT}) * (\text{the number of tokens to be generated})$
- **Throughput**: The number of output tokens per second an inference server can generate across all users and requests

Questions

- *In what scenario/use cases is TTFT (TPOT) more important?*
- *What is the latency-throughput tradeoff?*
- *What are the factors that affect different LLM performance metrics?*

(Nvidia) GPU and CUDA Primer



SOURCE: <https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/>

(Nvidia) GPU and CUDA Primer



SM



GA100 Full GPU with 128 SMs

Prefill vs. Decode Resource Demand

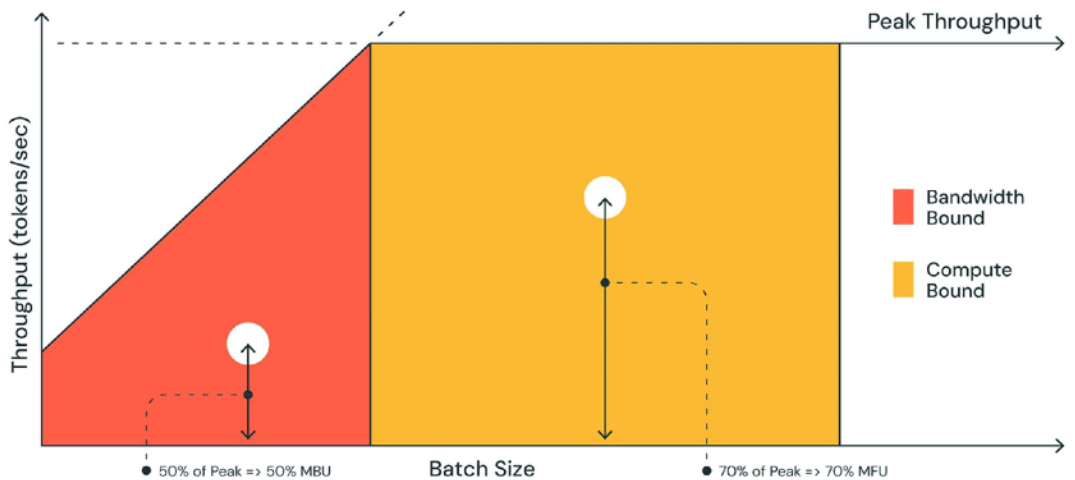
- GPU has two general types of computing resources:
 - GPU kernel and GPU memory
- Is prefill compute or memory bound?
- Is decoding compute or memory bound?

Common Techniques to Improve LLM Inference Perf

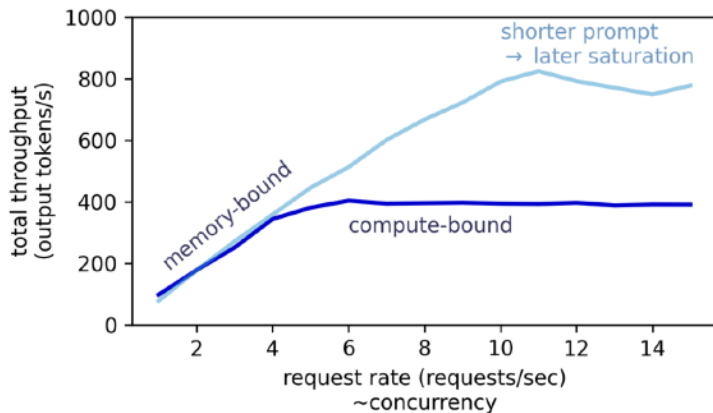
- **KV Cache**
- **Operator Fusion:** Combining different adjacent operators
- **Quantization:** Use fewer bits for weights and activations
- **Compression:** Sparsity or distillation
- **Parallelization:** Tensor parallelism across multiple devices or pipeline parallelism for larger models.
- More later this quarter

Model Bandwidth Utilization (MBU)

- $MBU = (\text{achieved memory bandwidth}) / (\text{peak memory bandwidth})$
- achieved memory bandwidth is $((\text{total model parameter size} + \text{KV cache size}) / \text{TPOT})$
- Model FLOPs Utilization (MFU) = $(\text{observed throughput (TPOT)}) / (\text{peak GPU FLOPs})$
- MBU important for memory-bound computation, MFU important for compute-bound computation. Ideally, want both to be 100%
- batch size: how many number of requests are sent to GPU for model forwarding at the same time (in parallel)



source: <https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices>



source: <https://huggingface.co/blog/tngtech/llm-performance-prefill-decode-concurrent-requests>

Observed MBU across various tensor-parallelized modes

*Higher is better

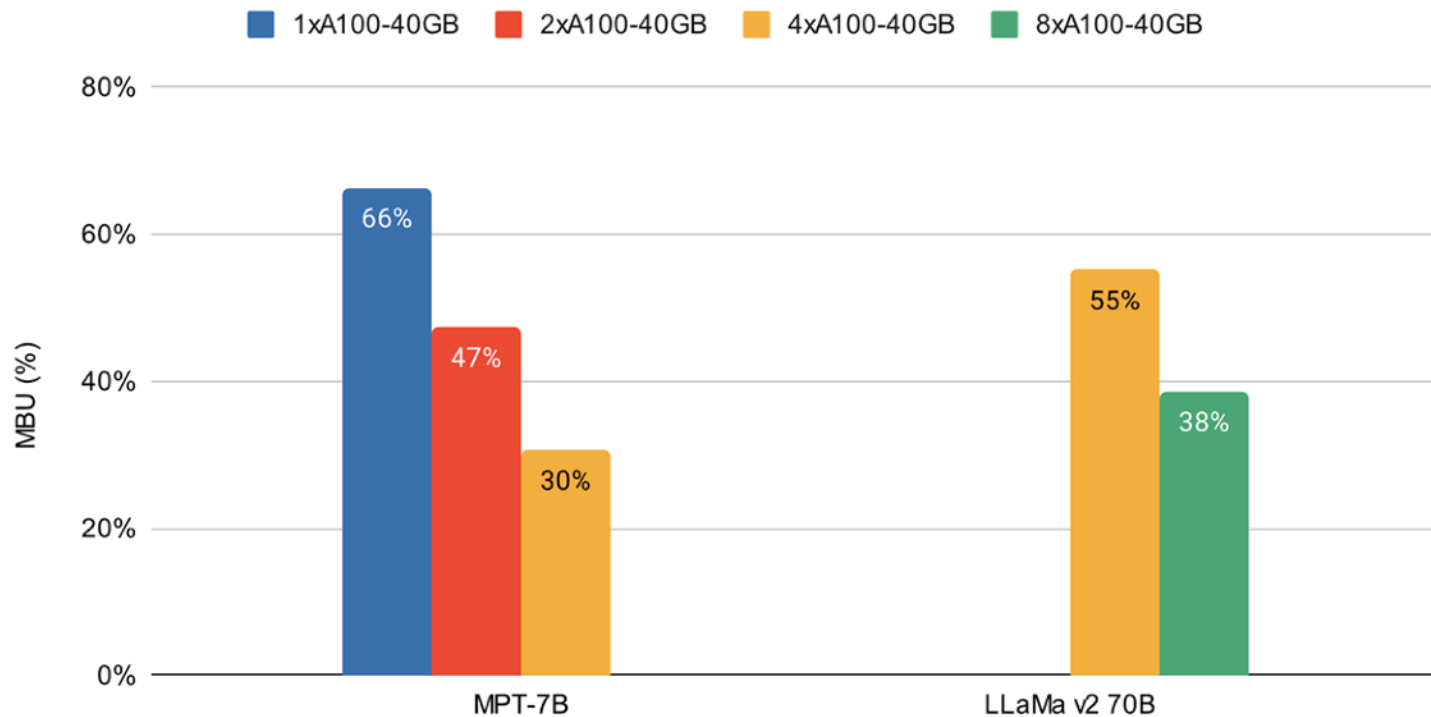


Figure 2: Empirically observed MBU for different degrees of tensor parallelism with TensorRT-LLM on A100-40G GPUs. Requests: sequences of 512 input tokens with a batch size of 1.

Observed MBU for varying batch sizes (Llama v2 70B fp16)

*Higher is better

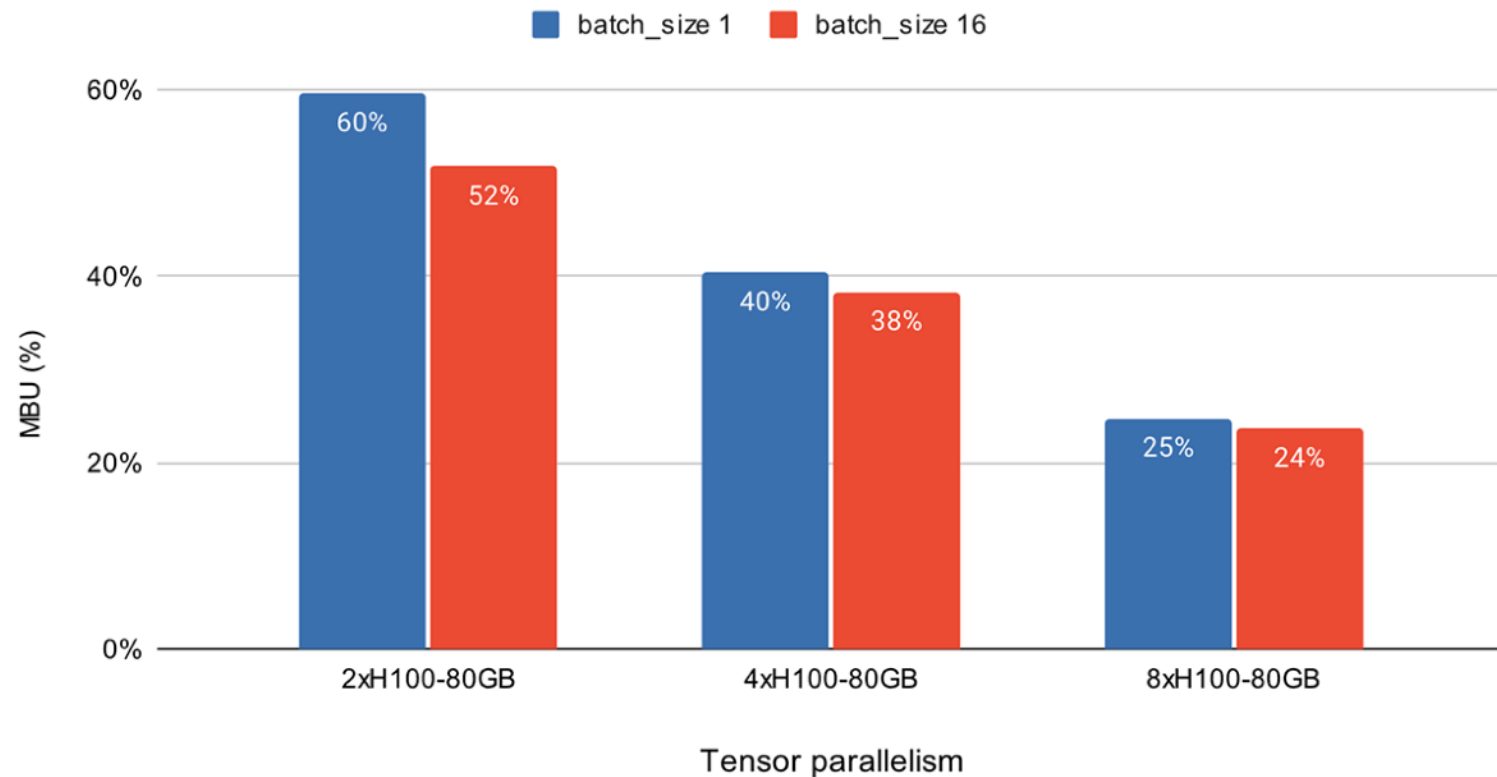


Figure 3: Empirically observed MBU for different batch sizes and tensor parallelism modes on H100-80G GPUs. Requests: sequences of 512 input tokens

source: <https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices>

MPT-7B: Throughput vs Latency

input tokens: 512, output tokens: 64

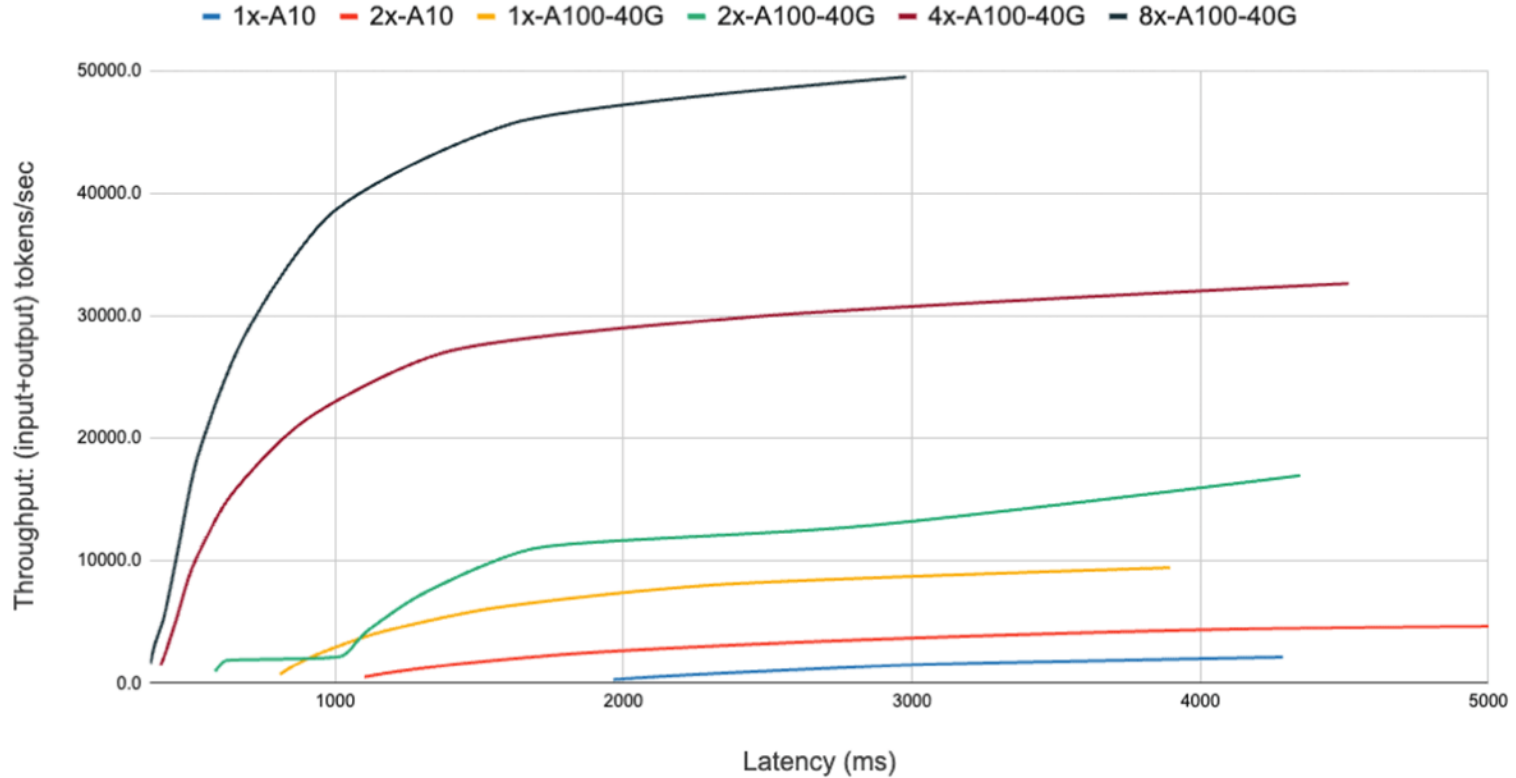


Figure 7: Throughput latency curve for the MPT-7B model. This allows users to pick a hardware configuration that meets their throughput requirements under a latency constraint.

source: <https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices>

Backup Slide: Google's TPU (Tensor Processing Unit)

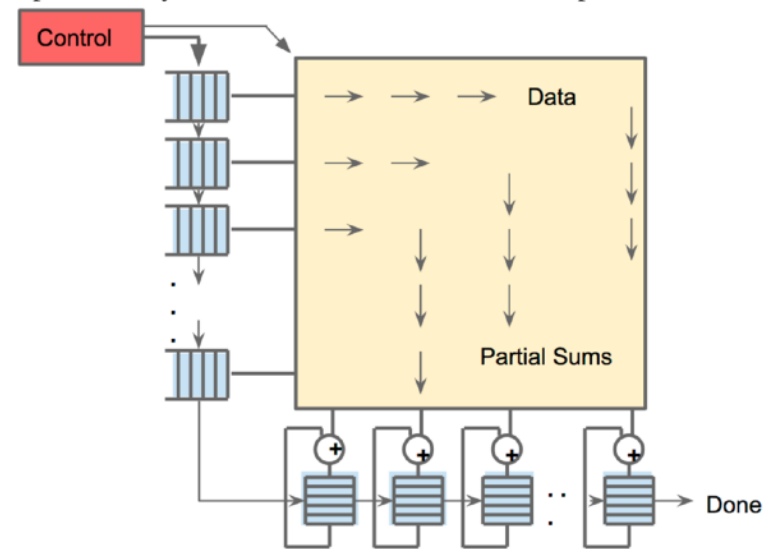
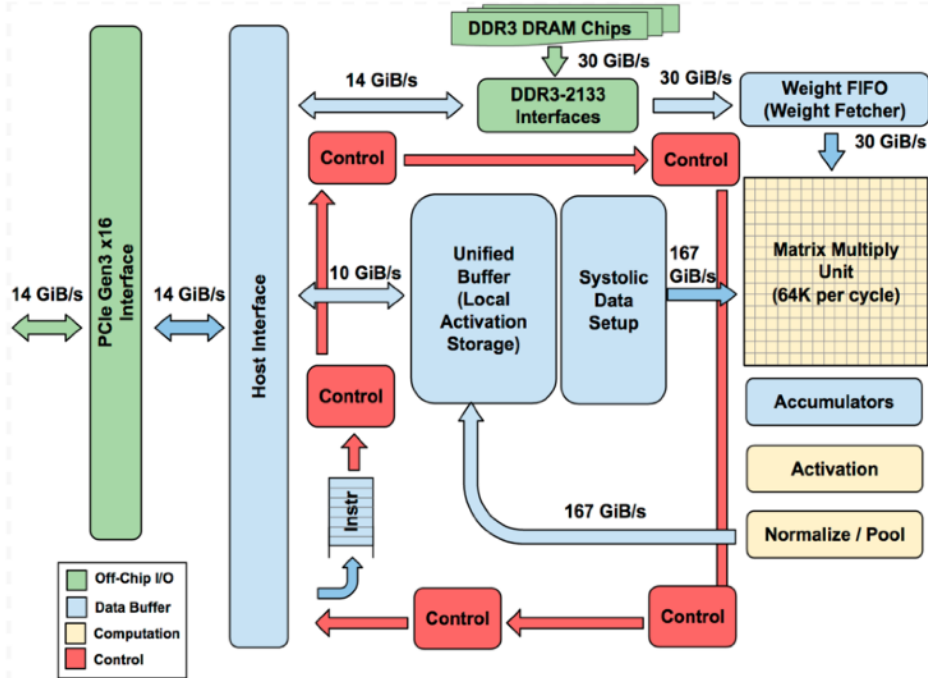


Figure 4. Systolic data flow of the Matrix Multiply Unit.