Performance of Transmormers

Arvind Krishnamurthy
Chien-Yu Lin

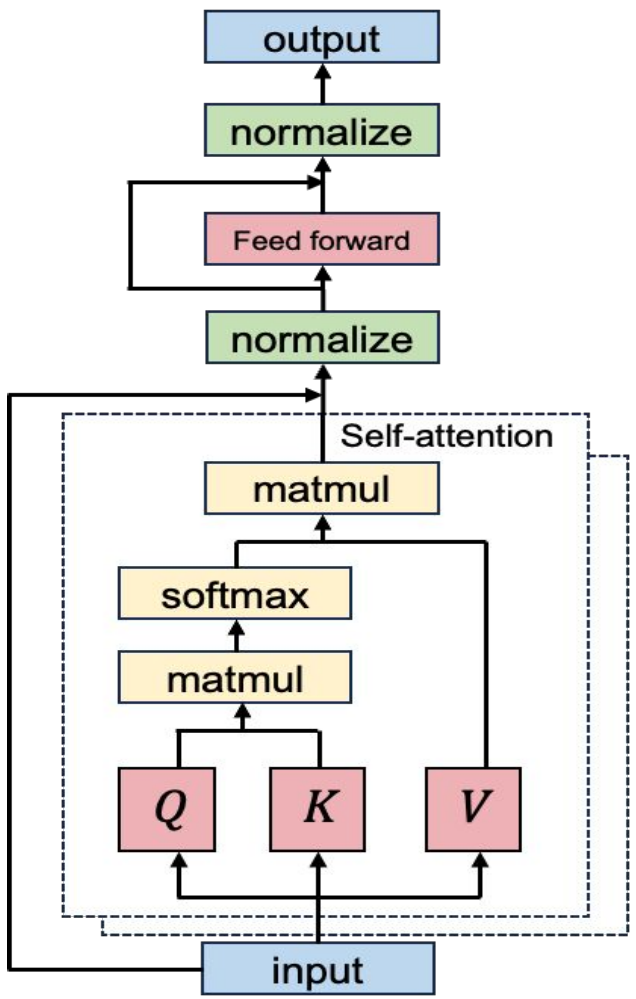
Material adapted from:
https://www.baseten.co/blog/llm-transformer-inference-guide
https://www.anyscale.com/blog/continuous-batching-llm-inference
https://arxiv.org/pdf/2402.16363

Lecture Outline

- Breaking down the components of a Transformer model
- Phases of a transformer model
- Dependencies in a transformer model
- Performance considerations
- Optimizations

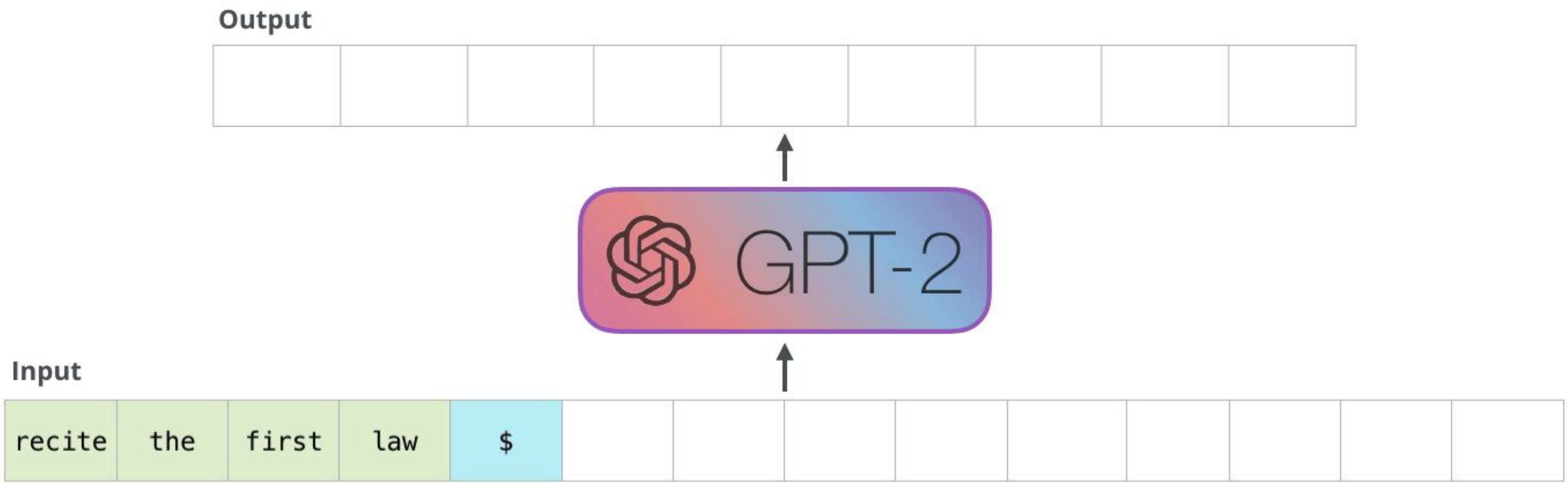
Transformer Model

- Two primary components:
  - Self-attention block
  - Feed-forward network block
- Two primary workloads:
  - Prefill workload (process the prompt, context)
  - Autoregression workload (token generation)



Prefill & Autoregression

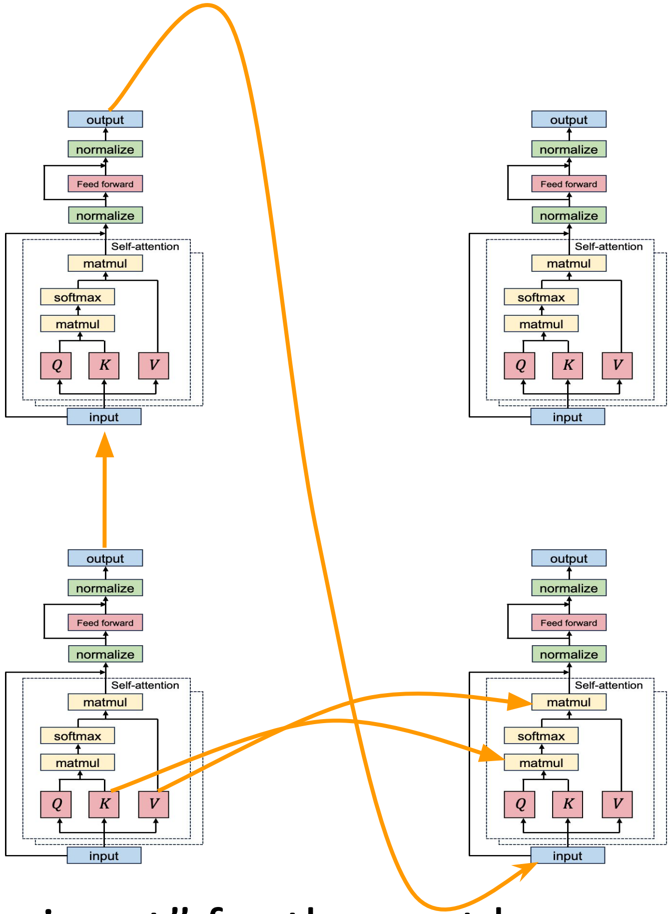
- Prefill: ingest all of the prompt & context that came with the query
- Autoregression: generate the response, one token at a time



State in a Transformer Model

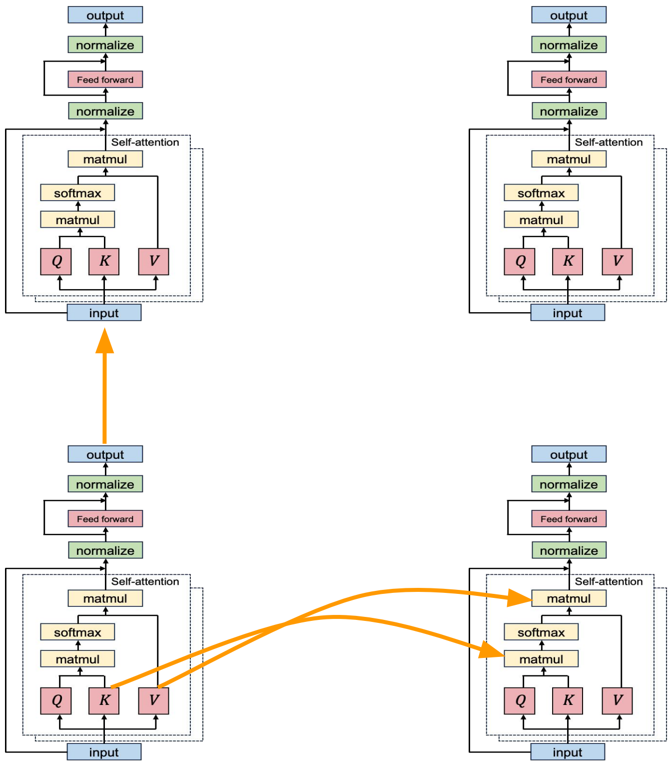
- For each token, at each layer of the transformer model:
  - Has an associated K, Q, and V vectors
- State can be regenerated as the computation is deterministic
  - But preferable to have this information be retained across autoregression steps
- Memory requirements of a transformer model:
  - Parameters of the model
  - K, Q, V vectors generated during inference

Dependencies for the Autoregression Step



- Output of each layer is the “token input” for the next layer
- K, Q, V values of prior tokens are required for subsequent tokens at each layer
- Output of the last layer is the generated token; subsequent autoregression step has a sequential dependency on this

Dependencies for the Prefill Step



- All tokens in the prompt/context are already available
  - Fewer dependencies and greater parallelism

What determines the performance of LLMs?

Transformer Performance

- Key considerations:
  - Compute requirements
  - Memory bandwidth of accelerator
  - Memory capacity of accelerator
  - For larger models, communication across GPUs
- Performance can be analyzed as the following cross-product
  - [Prefill, Autoregression] x [Perf. of Attention block, Perf. of FFN block]

Typical GPU Performance Parameters

- A10 GPU – slightly lower end, used more for inference than training

FP32	31.2 TF
TF32 Tensor Core	62.5 TF | 125 TF*
FP16 Tensor Core	125 TF | 250 TF*
INT8 Tensor Core	250 TOPS | 500 TOPS*
INT4 Tensor Core	500 TOPS | 1000 TOPS*
GPU Memory	24 GB GDDR6
GPU Memory Bandwidth	600 GB/s
Max TDP Power	150W

Key Accelerator Metric

- What is the balance between compute and memory?
  - Compute capability: 125TF
  - Memory bandwidth: 600GB/s
  - Ops/byte = 125TF / 600GB/s

    = 208.3 ops/byte
- GPU will be compute bound if we can do ~200 ops/byte
  - Else it will be memory bandwidth bound

Analyzing Compute/Memory Boundedness

- We will consider the Llama 2, 7B model
- Let us focus on just the attention block
  - Per-head dimension d, # heads = h [For Llama 2, d = 128, h = 32]
  - D = h*d
- Sequence length, N, of the input. Typical value = 4096
- FFN layers typically expand D to a larger size and project it down
  - Llama 2 expands D to an FFN size of 11008 and then projects it down to 4096

Analyzing Compute/Memory Boundedness

- Let us focus on just the attention block

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

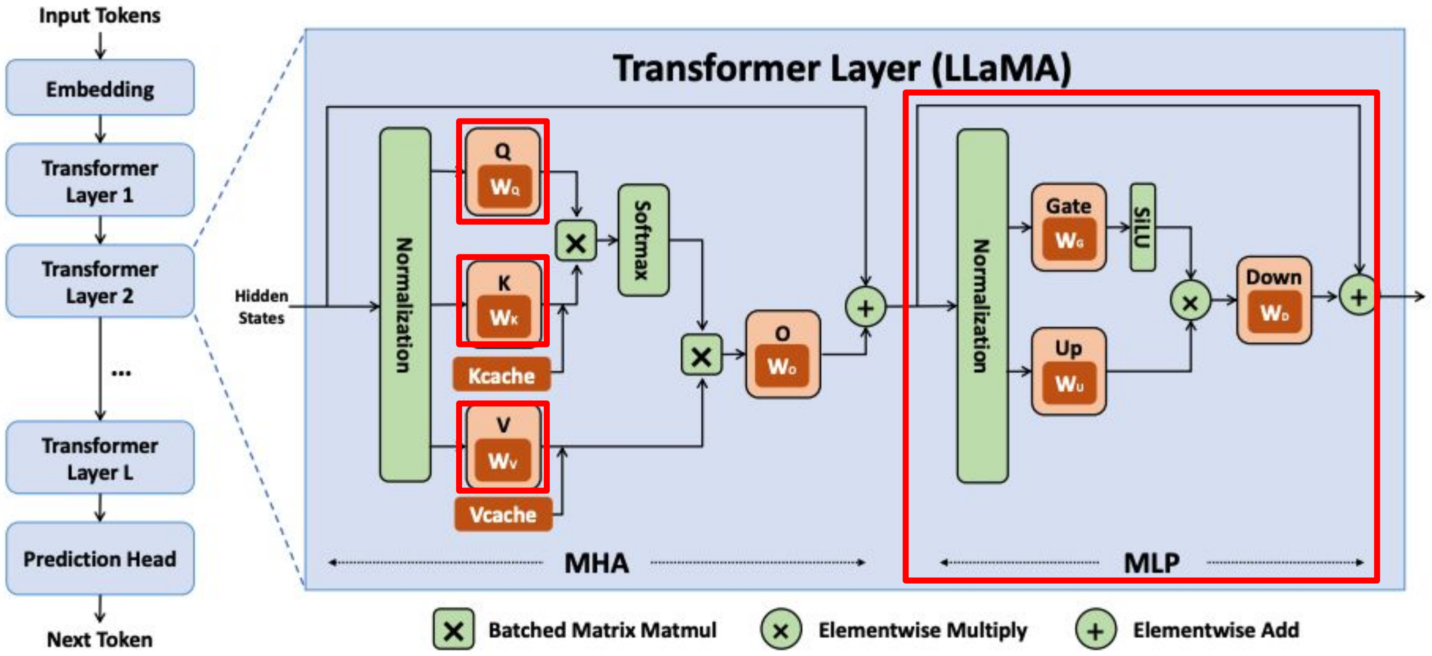
- We can calculate compute flops and memory loads/stores per head
  - Compute = 2*d*N*N + 3*N*N + 2*N*N*d
  - Loads/stores = 2*2*d*N + 2*N*N + 2*N*N + 2*N*N + 2(N*N + N*d) + 2*N*d

Analyzing Compute/Memory Boundedness

- Compute FLOPs/Memory ops = 62 ops/byte
- Significantly less than the desired ~200 ops/byte
- What is the underlying reason for this?
  - Compute = (4*d+3)*N^2
  - Loads/stores = 8*N^2 + 8*N*d
- How can we address this issue?

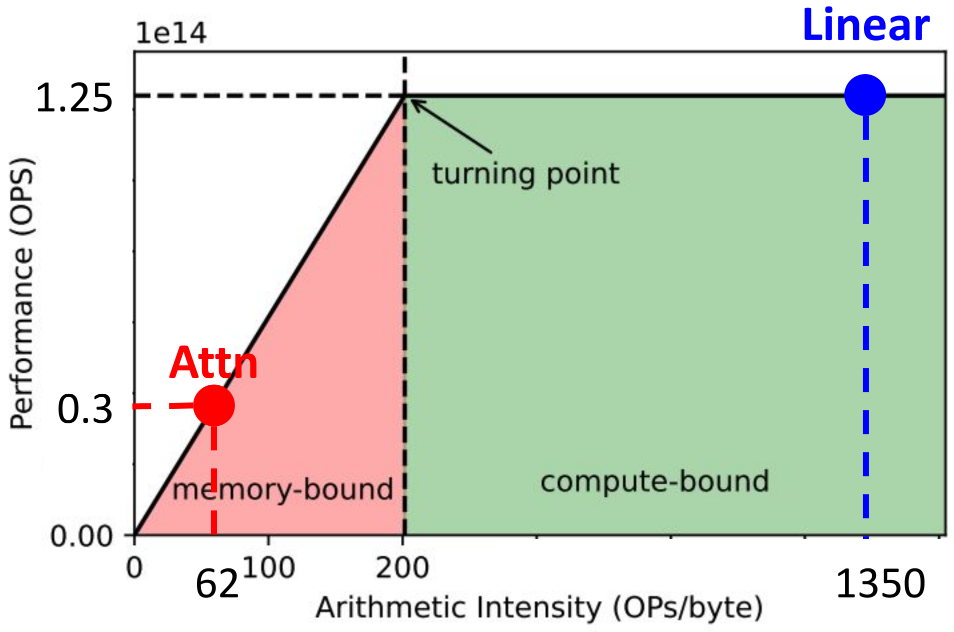
Analyzing Compute/Memory for FFN + Q,K,V

- Linear layer, essentially a GEMM: X * W
  - Shape: X (N, K), W(K, M)
- Compute: 2*N*K*M
- Memory: 2*N*K + 2*K*M + 2*N*M



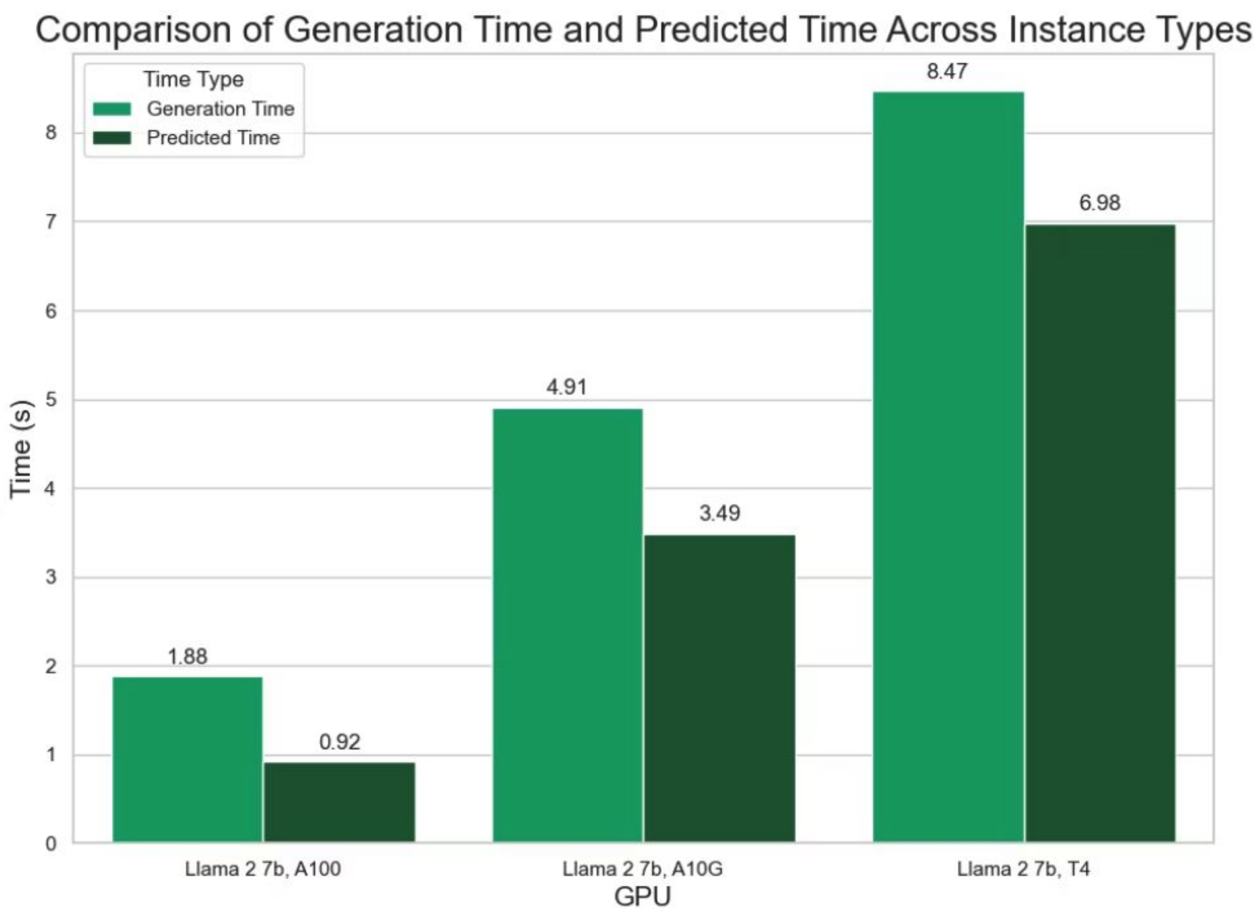
Analyzing Compute/Memory for FFN

- Compute FLOPs/Memory ops = 1365 ops/byte
  - N = K = M = 4096
- Much higher than the desired ~200 ops/byte
  - Compute bound
- How about decoding: N = 1?
  - Becomes GEMV, memory bound



Simple E2E Performance Model

- Assume that prefill is compute bound and decode is memory bound
- Execution time prediction = S*(2*#params/FLOPS) + G*(2*#params/MBW)
  - where S is prefill length and G is generated length



Batching to the rescue?

- Can we use batching to improve arithmetic intensity of attention?
  - Batch across tokens within the same request
  - Batch across tokens from different requests
- When is batching actually helpful?

Batching Optimization

- When a vectors can be utilized across different matmuls, then we can improve the arithmetic intensity
- Two scenarios where this reuse can take place:
  - One of the vectors in a matmul is a “model parameter”
    - Example: “$X*W_Q$, $X*W_K$, $X*W_V$, FFN layers
  - One of the vectors in a matmul is token state, but the same vector is involved in multiple operations with different token states
    - Example K of token 0 is interacted with Q of tokens 1, 2, 3, …

Batching Optimization

- We can now analyze the value of batching in the context of Prefill and Autoregression (aka decode) for Attention and FFN layers

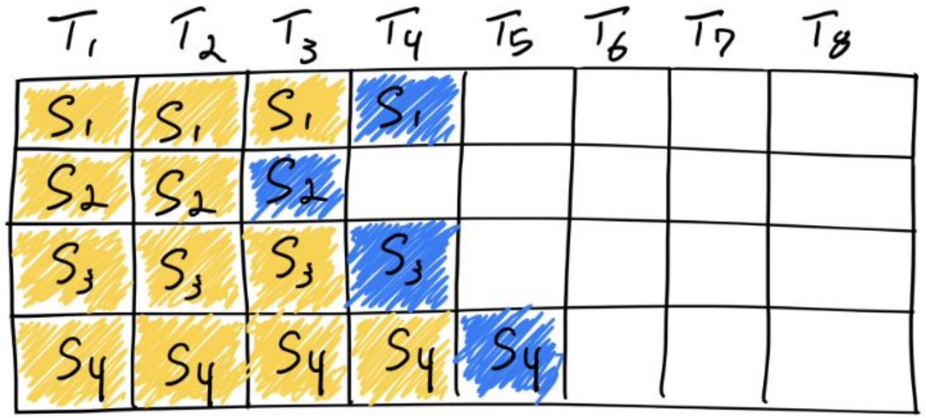
	Attention	FFN
Prefill	Batching useful	Batching useful
Decode	Batching across ops not useful	Batching across ops useful

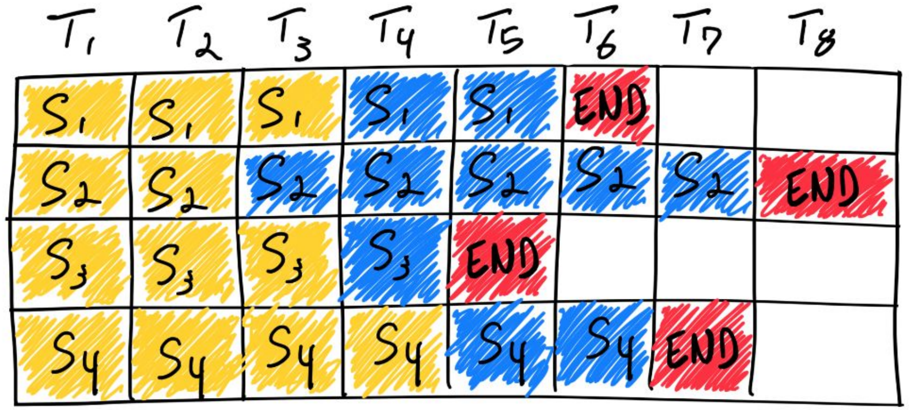
How much batching is possible?

- Depends on the memory available on the GPU
- GPU needs to accommodate parameters and kv-attention-state
  - KV-attention state = 2 * 2 * Num-layers * D * N
  - For Llama 2 with 4K tokens, KV-attention state is ~2GB
    - Parameter size is ~14GB
    - On A10, this means that we can have ~5 resident queries

Batching Implementation

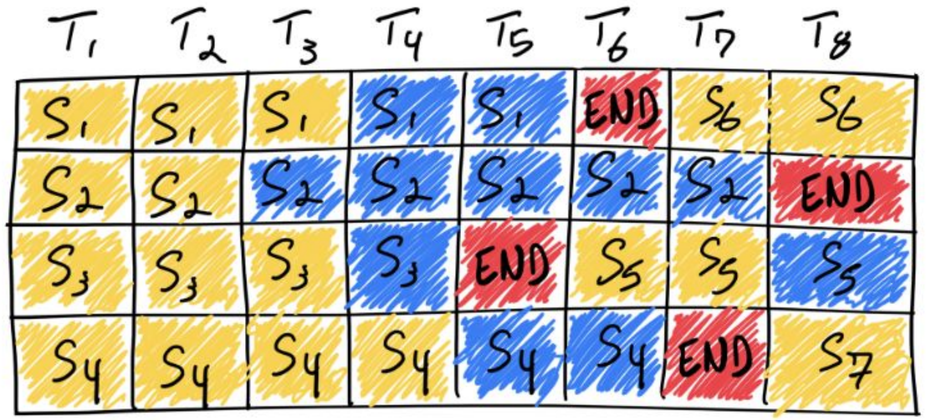
- Batching of autoregression isn't straightforward given heterogeneous requests
- Naive batching technique:
  - Accumulate "b" requests
  - Perform prefills for them in parallel (heterogeneity in costs with different lengths)
  - Start autoregression on b requests; wait for all requests to complete



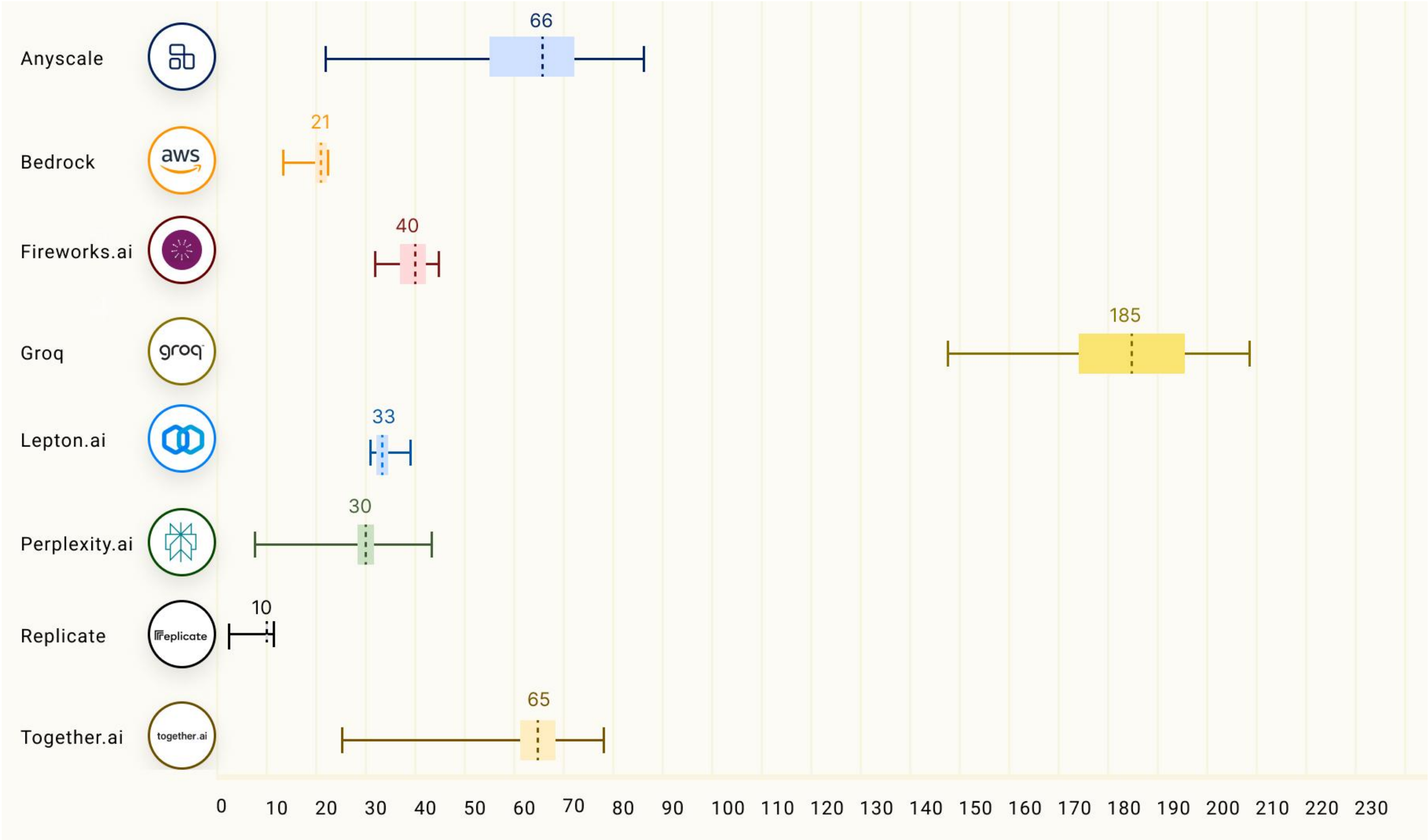


Continuous Batching [Orca - OSDI'22]

- Integrate new requests as old requests rotate off
- Need to perform prefill for new request
  - Results in a stall for existing requests
- Broader question, what are SLOs?
  - TTFT: time to first token requirement
  - TPOT: time per output token requirement
- Systems need to satisfy SLO requirements



LLM Performance leaderboard (tokens/sec)

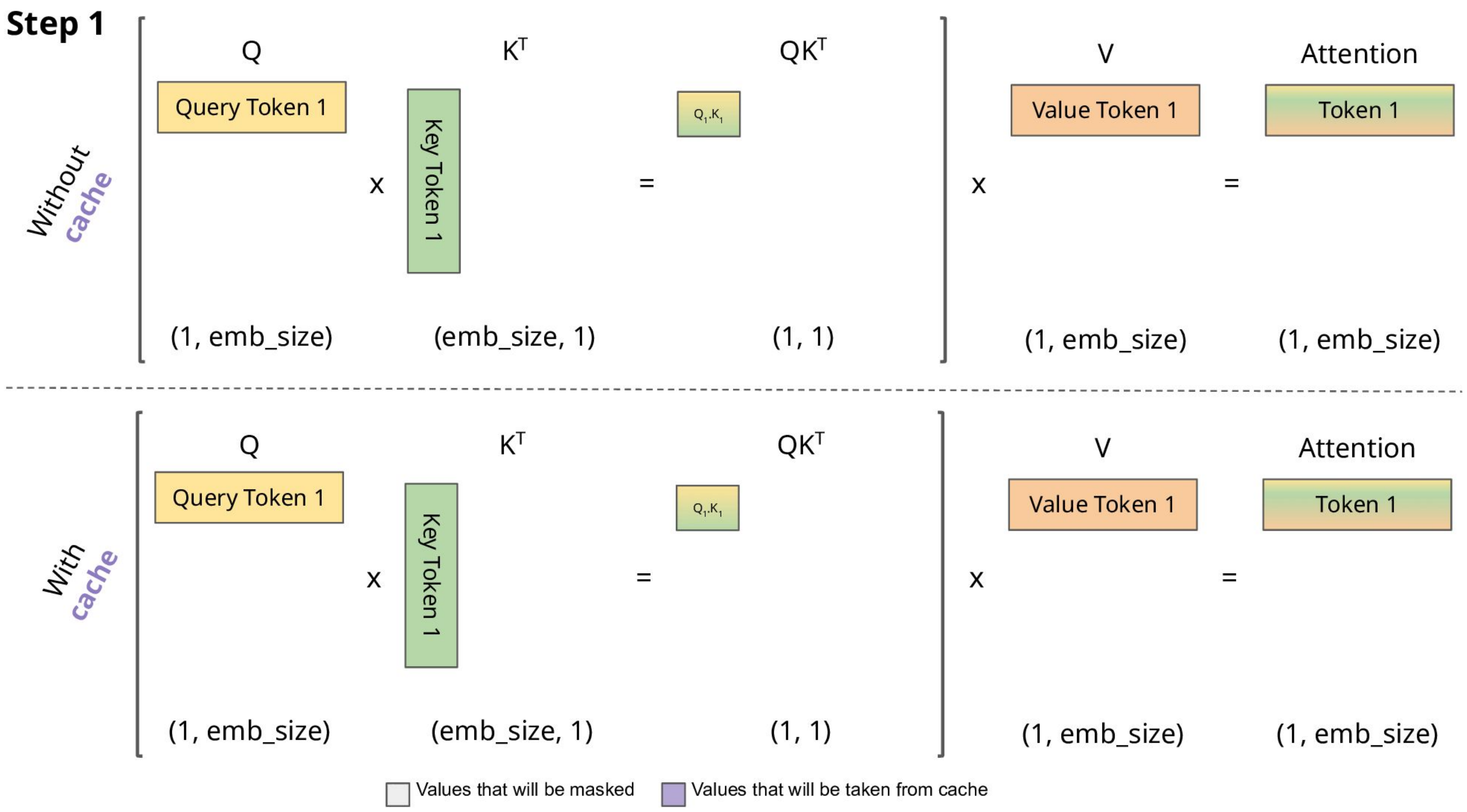


LLM Performance Optimizations

- KV Cache
- Mixture of Experts (MoE)
- Operation fusion
- Speculative decoding
- Quantization
- Pruning & Distillation
- Contextual Sparsity
- …

KV Cache

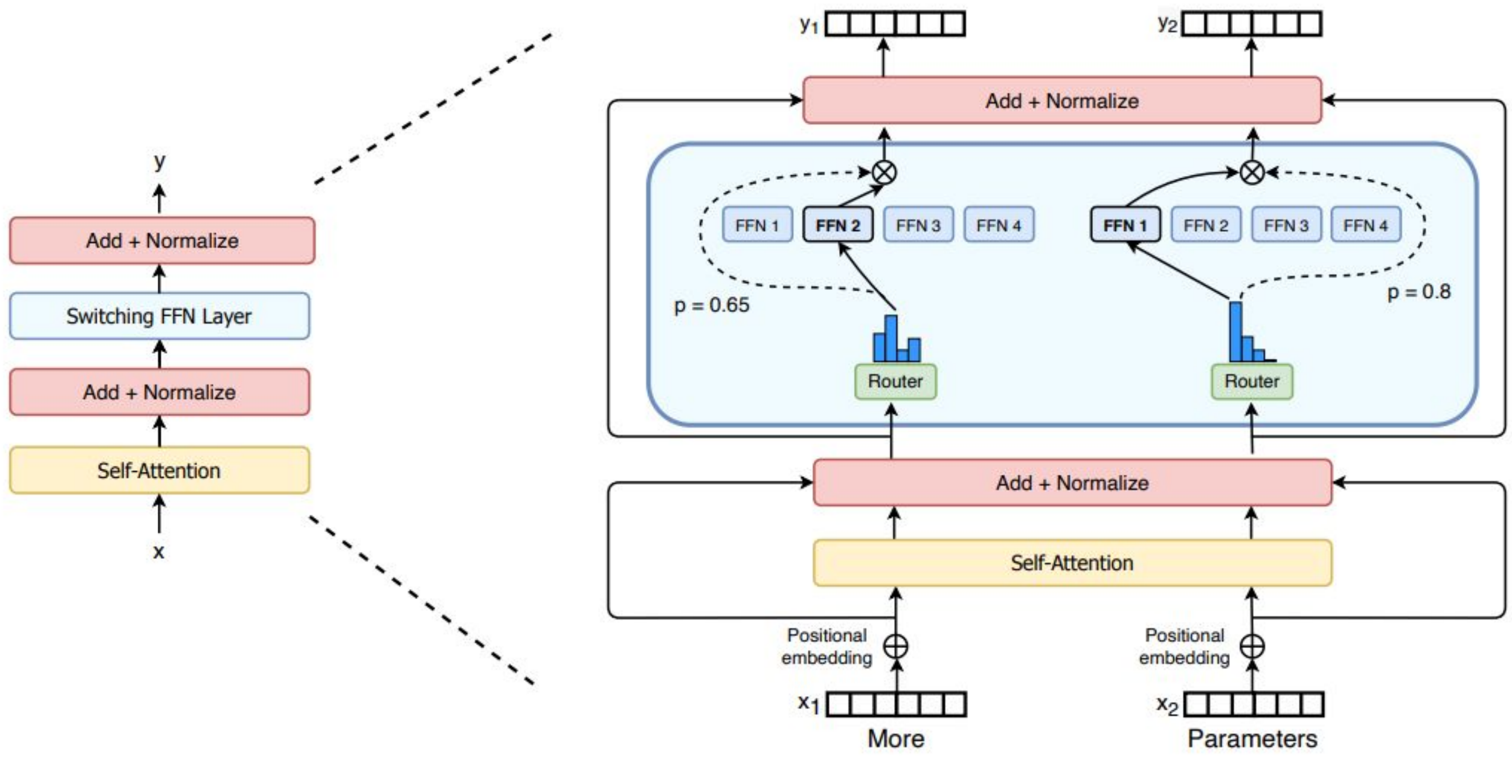
- Avoid recomputation of K and V for previous generated tokens



https://medium.com/@joaolages/kv-caching-explained-276520203249

Mixture of Experts (MoE)

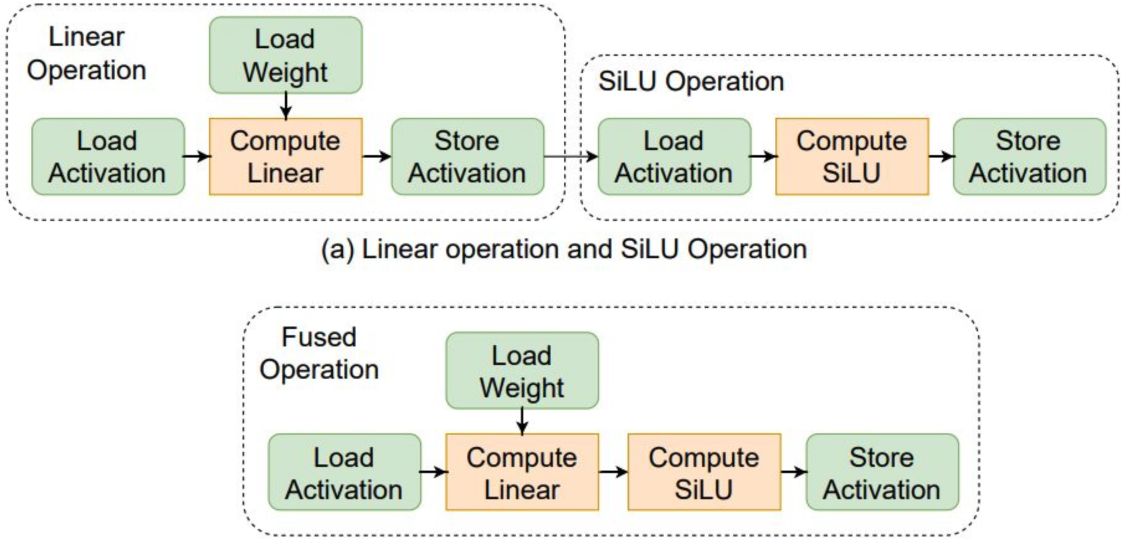
- Decouple computation and parameter counts for FFN
  - Keep inference FLOPs while increasing total parameters counts



https://huggingface.co/blog/moe

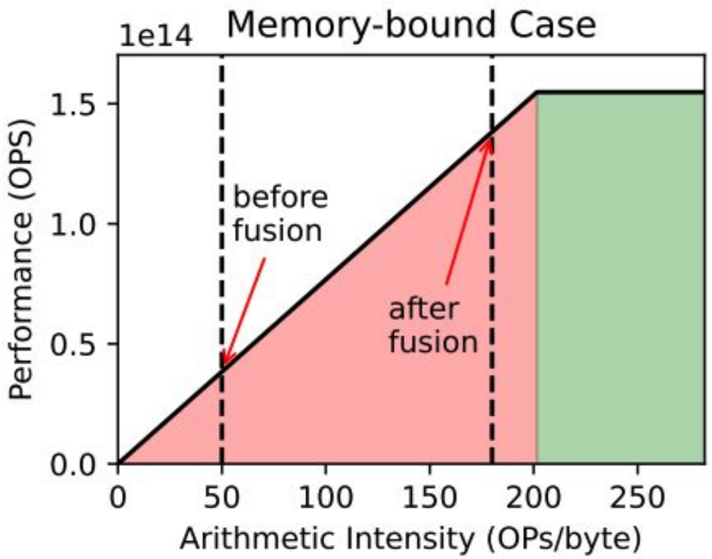
Operator Fusion

- Fuse neighbor operators on the computational graph
- Reduce memory movement on intermediate data
  - Intermediate data must not have dependencies to other ops
- Increase throughput for memory bounds ops



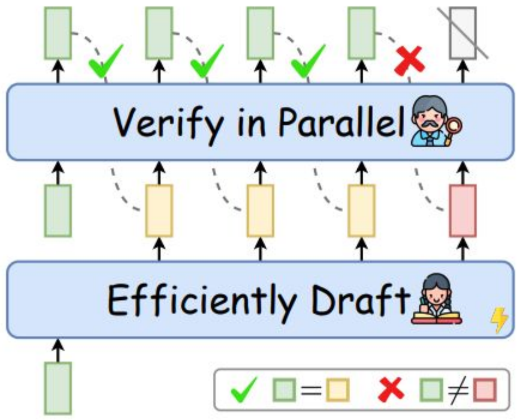
(a) Linear operation and SiLU Operation

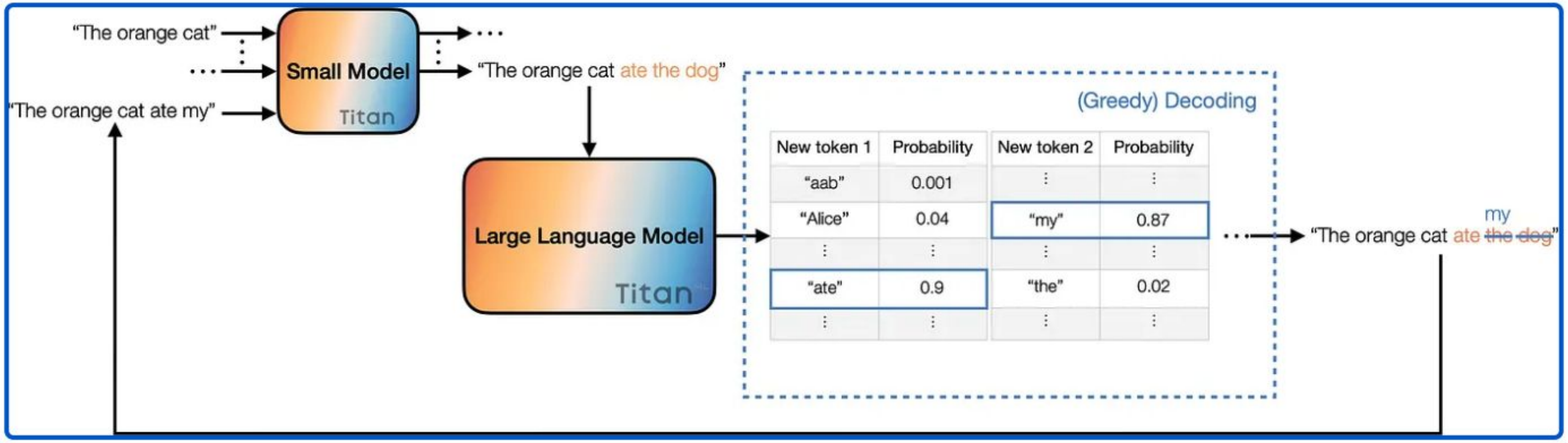
(b) Fused Operation



Speculative Decoding

- Predict tokens with small & fast models
- Verify with LLM to ensure generation quality
  - Verification is similar as Prefill, or “Append”





https://medium.com/@TitanML/in-the-fast-lane-speculative-decoding-10x-larger-model-no-extra-cost-f33ea39d065a

LLM Performance Optimizations

- KV Cache
- Mixture of Experts (MoE)
- Operation fusion
- Speculative decoding
- Quantization
  - Reduce size of each parameters
- Pruning & Distillation
  - Reduce number of parameters
- Contextual Sparsity
  - Skip tokens when decoding
- …