# Inference

最大化 GPU (memory + compute) 利用率 来提升 throughput 和降低 latency.

## I. Token Generation

目标:

1. 给定prompt及已生成的tokens, 每次通过 forward model 得到1个新token 的 logits.
2. 根据新 token 的 logits 概率分布, 通过采样 (sampling) 选取1个 token id.

**sampling (standard)**

调整 logits 的概率分布. 常见的方法 / 参数：

1. greedy (argmax): token id with largest logit.
2. top k: select from the top k token id with largest logits.
3. top p: select the top n tokens with cumulative probability >= p.
4. temperature: the denominator factor (T) of logits before softmax(); T++ -> random.

**pseudocode**

```
=========================================================
====
# forward LM to get next token
logits = model(prompt_tokens)
logits = logits[-1]   # dim: (T, V) -> (, V)

# apply temperature
logits /= temperature

# apply top_k
if top_k:
    v, _ = torch.topk(logits, top_k)
    logits = torch.where(logits < v[[-1], -float("inf"),
logits)

# calculate probability
probs = torch.softmax(logits, dim=-1)

# apply top_p
if top_p:
    sorted_probs, sorted_idx = torch.sort(probs,
descending=True)
    cumulative_probs = torch.cumsum(sorted_probs, dim=-1)
```

```
    sorted_idx_discard = cumulative_probs > top_p
    probs[sorted_idx[sorted_idx_discard]] = 0.0

# sample new token
new_token = torch.multinomial(probs, num_samples=1)
=========================================================
====
```

*pseudocode for token generation*

## II. Infra for Model Serving

**a. prerequisites**

prefill + decode

inference == 1 prefill step + n decode steps

- prefill: 根据 prompt 生成第1个 token; compute-bound.
- decode: 根据 prompt 和 已生成的 tokens 来生成下1个 token; memory-bound.

KV cache

将 {prompt + generated tokens} 在模型 attention 层 的 K, V vectors 存到GPU -> next decode step 可以直接调用; trade memory (-) for compute (+).

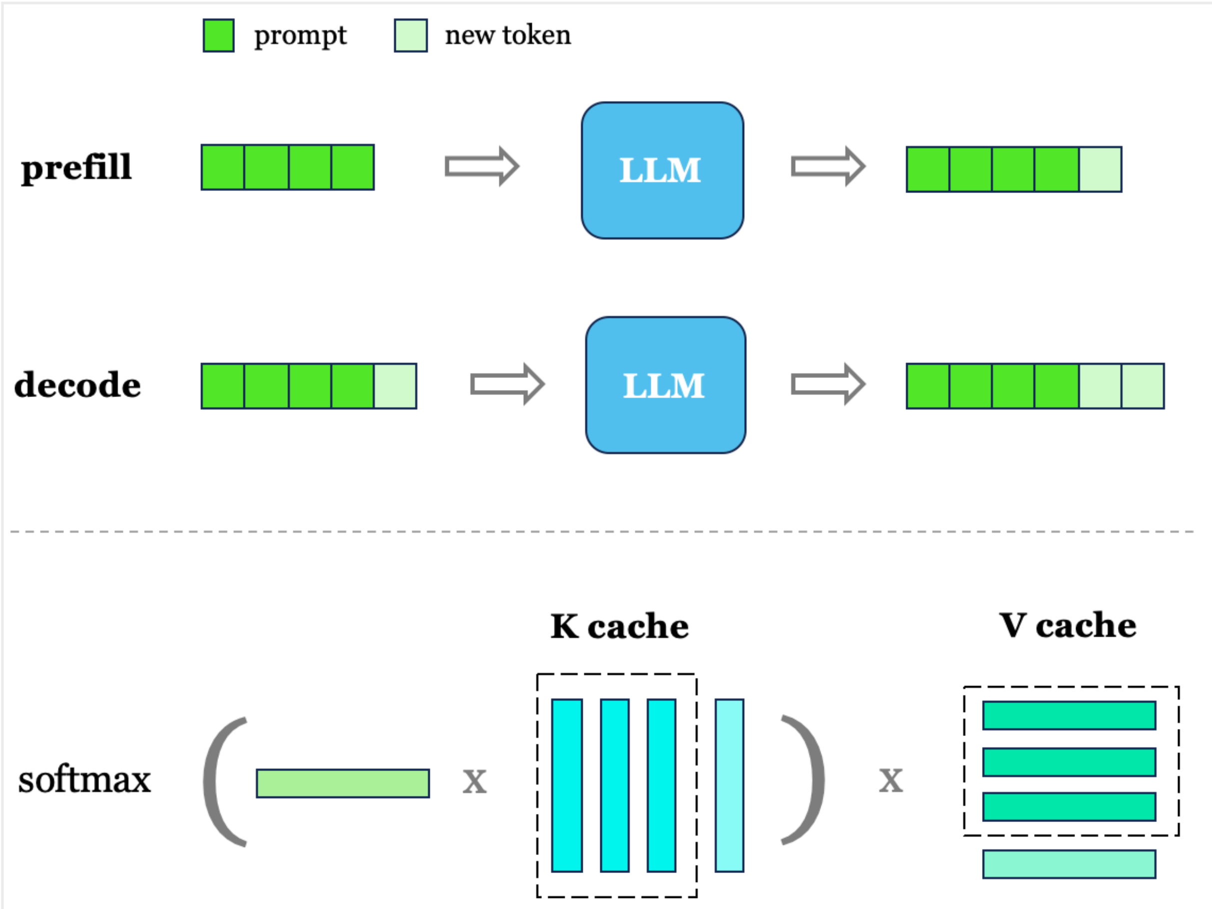


*Figure 1: prefill and decode; KV cache*

GPU memory allocation
model + KV cache + activation (intermediate output during the forward pass)

metrics / observability
- time-to-first-token (prefill)
- time-per-output-token (decode)

infra 底层逻辑
1. scheduler: plan and allocate available GPU (memory) for {prefill, decode} requests.
2. workers: execute {prefill, decode} task on GPU.

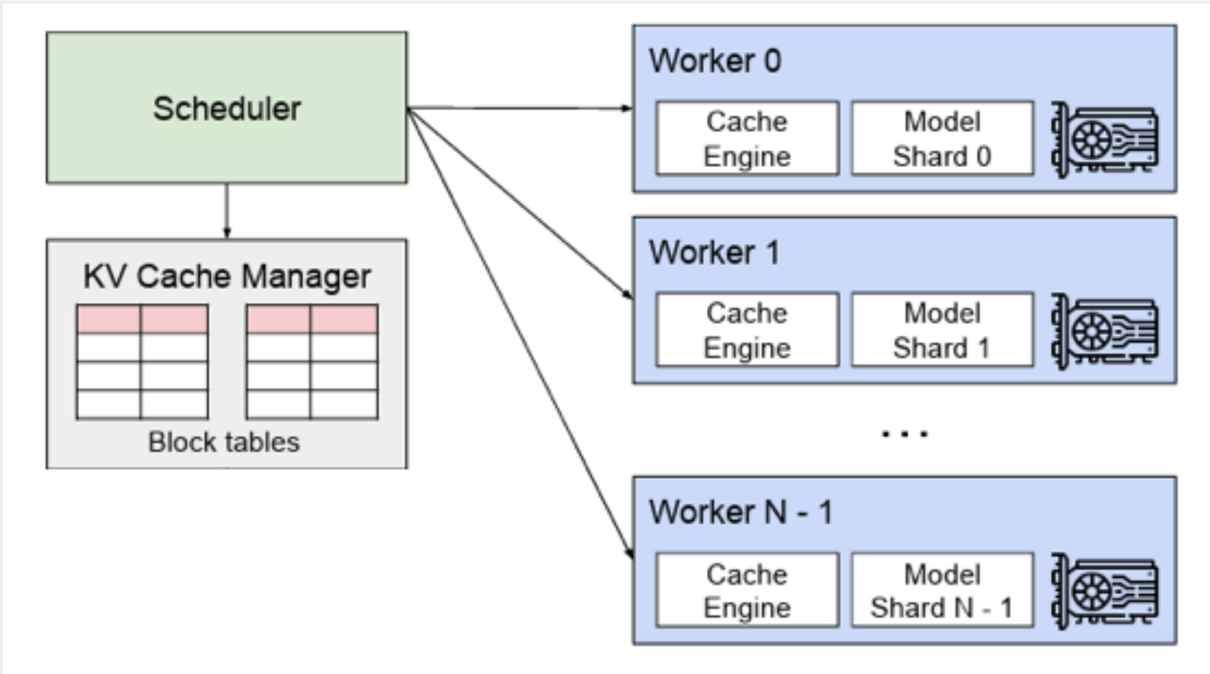


*Figure 2: overview of infra (from vLLM paper)*

**b. infra optimization**
优化角度: 1) 模型运算速度, 2) GPU 资源调度, 3) tradeoff between memory & compute

<u>batching</u>
- request-level: synchronous
  - pre-allocate GPU memory for each sequence -> memory waste
  - wait for the longest sequence to finish -> compute waste
- Iteration-level: asynchronous
  - schedule sequences to max out GPU capacity.

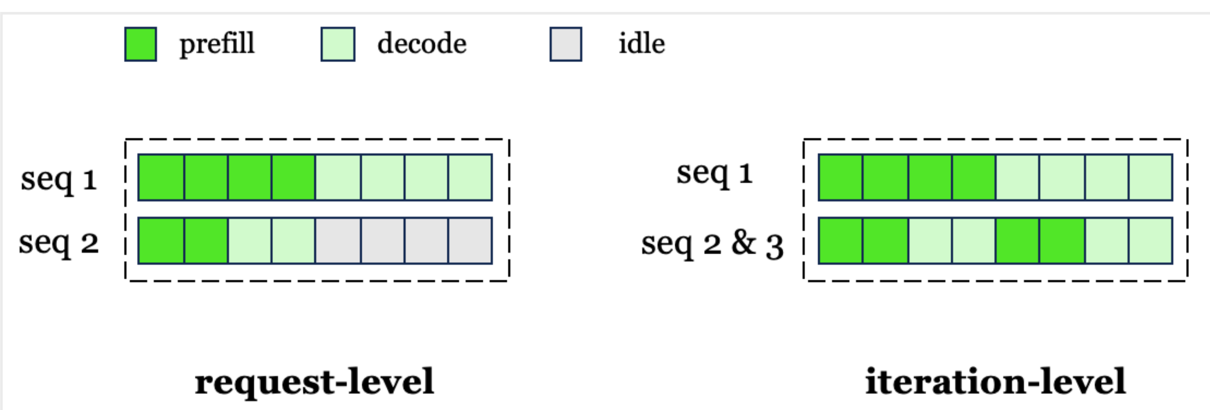


*Figure 3: request- and iteration-level batching*

KV caching (trade memory for compute)

- naive: allocate a chunk of contiguous memory for each sequence.
- PagedAttention (vLLM): divide cache into blocks -> reduce memory fragmentation.
- prefix caching: hash block with token prefix as key -> accelerate prefill with sys pmt.

scheduling

- vLLM: prefill-prioritizing, execute both the prefill and decode request as single step.
- chunked prefill: split prefill request into chunks & interleave with decode steps.
- prefill decode disaggregation: assign dedicated GPU for prefill and decode requests.

speculative decoding

- steps: 1) rollout n tokens using a small (fast) model 2) verify each token with the base model in parallel through reject sampling 3) accept the first k tokens.
- k > 1 -> reduced latency.

**c. frameworks**

- vLLM
- SGLang

## * Reference

generation

- lit-llama

infra

*blog & video*

- 猛猿 (zhihu): 691045737, 692540949
- 月球大叔 (YouTube): EP 1-6

*papers*

- Efficient Memory Management for Large Language Model Serving with PagedAttention
- Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve
- DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving
- Fast Inference from Transformers via Speculative Decoding