HW1

Tguo67

Question(s):

The submission due is 11:59pm ET on **Aug 26th.** The bug in canvas doesn't allow me to set the due date as it.

In this homework, you need to submit a pdf describing the workflow of vLLM based on its class hierarchy in its document.

Specifically, when a request comes, how is it processed through this class hierarchy?

Related links:

- https://docs.vllm.ai/en/latest/design/arch_overview.htmlLinks to an external site.
- https://github.com/vllm-project/vllmLinks to an external site.

Answers:

1. HTTP layer (OpenAI-compatible server): FastAPI routes the call into vLLM's OpenAI server handlers, and the related auth, payload validation, and sampling params are parsed.
2. Async engine client & IPC boundary: tokenizes/detokenizes on the API side and sends the work to the core engine process over async IPC (asyncllm).
3. Engine core process: loop iteration ("engine step") does:
   a) pull new/continued work from its input queue,
   b) ask the Scheduler what to run now (continuous batching),
   c) marshal inputs and call the ModelExecutor,
   d) push token results to its output queue (which the API side reads/streams).
4. Scheduling & batching: tracks all active requests / sequences and decides, per step, how many tokens each should advance under a token budget. This is the heart of continuous batching (a mix of chunked prefill + decode in one pass).
5. Memory management for attention (KV cache): hands out fixed-size KV blocks for each request/layer, so the engine can reuse attention states efficiently across steps and between requests (PagedAttention).
6. Model execution fan-out: coordinates one (single-GPU) or many workers (multi-GPU) and ships the per-step batch to a worker process. Each worker hosts a ModelRunner (GPU runner) that actually loads the model and performs the forward pass.
7. Sampling & outputs: The engine applies sampling (temperature/top-p/etc.) and forms new tokens for each active sequence.
8. Done / book-keeping: When stop conditions are met (e.g., EOS, token limit), the engine finalizes the request, frees KV blocks in KVCacheManager, and the server sends the final payload.