Nano-vLLM Execution Flow

Step 1: Initialization & Resource Allocation

Before any inference happens, the engine sets up the environment:

- **Initialize Components:** Load the Tokenizer, the Model (LLM), and define SamplingParams (temperature, top_p, etc.).
- **Block Manager Setup:** Initialize the BlockAllocator which manages the mapping between **Logical Blocks** (what the sequence sees) and **Physical Blocks** (actual GPU memory slots).

Step 2: Request Submission

- **Encoding:** The input prompt is tokenized into a list of token_ids.
- **Sequence Group:** A SequenceGroup object is created to track the state.
- **Queueing:** The request is placed into the **Waiting Queue**. At this stage, no GPU memory (KV cache) is allocated yet.

Step 3: The Iteration Loop (step() function)

The engine enters a loop, calling the step() function repeatedly until all sequences are finished.

A. Scheduling

The Scheduler decides which sequences to process:

1. **Prefill Priority:** It pulls sequences from the **Waiting Queue**.
2. **Resource Check:** It checks if there are enough free physical blocks to accommodate the prompt.
3. **State Transition:** Sequences move from Waiting to Running.

B. Execution: Prefill Phase (Initial Prompt)

For new sequences:

- **Block Allocation:** The Block Manager allocates enough physical blocks to store the prompt's KV keys/values.
- **Model Run:** The model processes the entire prompt in one "chunk."
- **Output:** Generates the **first** hidden state and samples the first token.

C. Execution: Decode Phase (Token-by-Token)

For sequences already in the Running state:

1. **Preemption Check:** If the GPU is out of memory for new tokens, the scheduler may "evict" a sequence (moving it back to Waiting).
2. **Slot Mapping:** To tell the GPU where to write the new token's KV pair, we calculate the exact memory offset:
   slot_mapping = (block_table[block_idx] * block_size) + block_offset

3. **CUDA Graph Execution:** Because decode kernels are small and frequent, nano-vLLM uses **CUDA Graphs** to "record" the GPU operations, eliminating the CPU launch overhead for every single token.
4. **PagedAttention Kernel:** The model runs, using the slot_mapping to retrieve past KV blocks and append the new token's KV data to the correct physical location.
5. **Sampling:** The next token is generated and appended to the sequence.