CHAPTER 10 / 4 MINUTE BRIEF

Specialized Hardware

Read the full chapter → / Read this brief online →

The main idea

A hosted model is not the only way to serve an AI workload. Data-placement requirements, latency or sustained volume may justify running a model on rented or owned GPUs, but none of them by itself requires buying hardware: a permitted regional service may solve the placement problem, and a negotiated API arrangement may solve the price problem. Compare those options before taking on a device fleet.

GPU capacity can be metered by instance time, reserved under contract or owned, and each choice changes lead times and operating obligations. The plan needs to know where the devices can run, how much power and cooling they need, which failures remove a whole group at once, and the supplier's current lead time; "measured in quarters" is not a property of every purchase.

Explore this section in the full chapter →

The arithmetic a planner owns

The raw weight payload is parameters times bytes per parameter. A seventy-billion-parameter model therefore needs about 140 GB at 16 bits, 70 GB at 8 bits or 35 GB at 4 bits, using decimal gigabytes. Actual allocation also includes scales, metadata and runtime state. Quantization can reduce memory and alter quality or speed; evaluate the chosen method rather than assuming a fixed quality loss.[1]

For a conventional transformer attention cache, the bytes per retained token are:

$$\begin{aligned}\text{cache bytes per token} = 2 \times \text{layers} \times \text{key-value heads} \\ \times \text{head dimension} \times \text{bytes per value}.\end{aligned}$$

[1] vLLM, "Quantization", checked October 3, 2026. Methods and supported hardware differ by release. Raw weight arithmetic excludes scales, metadata and runtime allocations.

The factor of two counts keys and values. Grouped-query attention can use fewer key-value heads than query heads, and other attention designs need different calculations. PagedAttention addresses cache allocation and sharing; it does not imply that every seventy-billion-parameter model has the same cache size.[2] The memory-only concurrency bound is:

$$\text{memory-only concurrency bound} = \frac{\text{device memory} - \text{weights} - \text{reserve}}{\text{KV cache per request}}.$$

Take a hypothetical configuration with 80 layers, 8 key-value heads, head dimension 128 and two bytes per cached value. The cache per token is $2 \times 80 \times 8 \times 128 \times 2 =$ **327,680 bytes**. At 8,000 retained tokens, a request needs 2,621,440,000 bytes, or about **2.62 GB**, before allocation overhead. Suppose 70 GB of weights fit in 141 GB and we reserve 14.2 GB for other use: $141 - 70 - 14.2 =$ **56.8 GB** remains. The memory bound is $\left\lfloor \frac{56.8}{2.62144} \right\rfloor =$ **21 requests**. With 64 key-value heads instead of eight, each cache would be eight times larger and the bound would fall to two. These invented configurations show why head count belongs in the calculation, not whether either workload meets its latency target.

Explore this section in the full chapter →

What must be measured

There is no universal factor that turns a month's token count into the number of GPUs to buy. The conversion requires a measured configuration, task distribution, latency requirement, availability arrangement and demand pattern. A serving team may know that long contexts change the bottleneck; a platform team may know that the required devices cannot be placed together by launch. Establish those conditions before negotiating a price.

Explore this section in the full chapter →

[2] Woosuk Kwon and colleagues, "Efficient Memory Management for Large Language Model Serving with PagedAttention," SOSP 2023, https://arxiv.org/abs/2309.06180. Describes the cache memory lost to reservation and fragmentation under contiguous allocation, and paged allocation as the remedy. Its OPT-13B example applies the same per-token formula: $2 \times 5{,}120 \times 40$ layers $\times$ 2 bytes, about 800 KB.

An experiment on shared hardware, before the fleet

The chapter opens on the arithmetic a purchased fleet needs, and that arithmetic starts with a serving experiment. The experiment itself does not need purchased or reserved hardware. Three shared services cover the step: an interpretability service that runs intervention code on models already deployed on shared GPUs, and two rented options — a GPU pod billed by the second, and a function platform that runs custom code on a GPU it selects. Both rented options charge for the time the work uses.

Explore this section in the full chapter →

The account and the meter teach the same lesson Chapter 2 teaches about elasticity: per-second or per-hour capacity keeps an experiment's cost proportional to the experiment. The fleet decision still needs everything this chapter sizes — goodput, failure groups, usable fraction — but the evidence arrives first, on someone else's capacity.

Explore this section in the full chapter →

The sizing process

Measure, model, decide, and come back afterward: the loop Chapter 4 ran for web workers. The serving experiment establishes real rates at the real configuration. The model connects demand to devices: concurrency, cache budget, latency targets, growth. The decision compares purchasing arrangements, failure conditions and lead times before committing to supply.

Explore this section in the full chapter →

The process should leave a record that someone else can check and rerun:

- **Workload:** peak arrival rate, prompt and output lengths, TTFT and TPOT objectives and the share that must meet them.
- **Configuration:** model, precision, engine version, device, VM size, devices per replica.
- **Evidence:** benchmark command and date, goodput per device, memory bound, bandwidth floor.
- **Failure and supply:** purchase unit, failure allowance, maintenance plan, supply arrangement, lead time, fallback.
- **Revisit when:** the model, precision or prompt lengths change, goodput falls, or a contract date nears.

Explore this section in the full chapter →