QUEUEING-THEORETIC GPU PROVISIONING FOR DISAGGREGATED MULTIMODAL LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Disaggregated multimodal LLM serving separates modality encoding from prefill and decode, but its throughput depends on how GPUs are divided between the two stages. We develop a queueing-theoretic framework for choosing this allocation under a fixed GPU budget. The main modeling challenge is estimating encoder service demand: CPU and GPU work overlap across requests, so neither GPU execution time nor isolated request latency determines sustainable throughput. We decompose the encoder path into resource demands, fit their dependence on visual token count, and identify the bottleneck that limits request completions. A theoretical two-stage tandem queueing model then yields a closed-form allocation rule and explicit rate intervals within which each integer allocation remains optimal. The rule supports initial provisioning and can provide allocation targets for runtime adaptation. We study the encoder pipeline on an RTX 4090 D testbed and evaluate provisioning in vLLM on eight H800 GPUs. Using a single measured pair of stage rates, the model predicts capacity within 6% across five tested allocations and selects the best measured split. Compared with an equal split, the computed allocation raises saturation throughput from 59.9 to 86.0 requests per second, a 43.6% improvement without additional GPUs.

1 INTRODUCTION

Multimodal models place a modality encoder in front of prefill and decode, converting raw inputs into tokens processed with the text prompt (Liu et al., 2023; Bai et al., 2025; Chen et al., 2024; Xu et al., 2025). Building on the disaggregation of prefill and decode (Zhong et al., 2024; Qin et al., 2024), encoder-prefill-decode (EPD) disaggregation assigns these phases to dedicated instances and is now both studied in the literature (Pan et al., 2025; Qiu et al., 2025; Singh et al., 2025) and supported in vLLM (vLLM Team, 2025). It creates a consequential planning decision: with a fixed budget, every GPU assigned to encoding is unavailable to the language model.

No universal E:PD ratio applies because each stage's work per request depends on the hardware, model, and workload. The penalty is first-order: in our eight-GPU experiments (Section 5), allocations on identical hardware range from 45.5 to 86.0 requests per second, and the symmetric 4:4 split leaves roughly 30 percent of achievable capacity unused. If the two sustainable stage rates were known, the solution would simply balance their aggregate capacities. The reason this has not been operationally trivial is that the encoder rate is easy to measure incorrectly. Its CPU preprocessing, scheduling, GPU execution, and handoff occupy different resource lanes and overlap across requests; isolated latency adds time that can overlap, while GPU time omits CPU bottlenecks. Provisioning therefore requires bottleneck resource occupancy under concurrency, not either familiar proxy.

Existing systems use workload measurements and runtime feedback to adapt resources (Wu et al., 2024; Jin et al., 2024; Li et al., 2025; Lai et al., 2025; Chen et al., 2025). These mechanisms remain valuable when demand changes. Role changes can temporarily remove workers from service while models or caches are reconfigured (Singh et al., 2025). We study a complementary planning question: which E:PD split should be deployed for a specified workload? Prior work also supports planning through offline profiling and configuration search (Qiu et al., 2025; Singh et al., 2025). Our approach computes the target directly from calibrated encoder resource demands and PD capacity estimates, without deploying candidate splits. The target can be supplied before deployment or updated from traffic logs, while runtime controllers execute required switches and respond to unexpected changes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Nie et al. (2026) derive KV-cache-aware stability bounds that guide PD sizing for a given model, hardware, request distribution, and arrival rate. We complement this PD-side analysis by estimating encoder demand and dividing a fixed GPU budget between E and PD. Our contributions are:

1. **A queueing-theoretic provisioning framework.** We model the E and PD pools as an effective tandem system. Given stage rates, the capacity-optimal integer split is obtained by comparing two candidates (Sections 3.1 and 3.2).
2. **Decision robustness and sufficient latency guarantees.** We derive rate-ratio intervals for each optimal split and a sufficient TTFT-feasible rate boundary under uniform tail bounds. We characterize when the certified TTFT objective retains the capacity-optimal allocation (Sections 3.3 and 3.4).
3. **A profiling methodology for encoder demand.** We decompose the encoder path into resource lanes, select functional forms from their mechanisms, and account for overlap and parallelism to estimate encoder capacity from resource loads (Section 4).
4. **Complementary methodology and scale-up evaluation.** A four-GPU experiment evaluates the demand-to-allocation pipeline. Separately, on eight H800 GPUs, one independently measured rate pair predicts all five tested capacities within 6% and selects the best measured split. Relative to the equal split, saturation capacity rises by 43.6%, and the highest tested SLO-compliant offered rate rises by 50% (Section 5).

Algorithm 1 recomputes the target from traffic logs while the calibrated demand models remain valid.

2 Background: EPD Serving and the Encoder Pipeline

EPD disaggregation. A multimodal request contains text and media, such as images. The encoder (E) converts the media into embeddings for the language model. Prior work separates encoding, prefill, and decode to reduce interference and scale the stages independently (Singh et al., 2025; Qiu et al., 2025). Figure 1(a) shows the configuration studied here: E and PD instances use separate GPUs, while each PD instance runs both prefill and decode. We use vLLM's existing encoder disaggregation (vLLM Team, 2025). A proxy sends the media to E. Once the embeddings are available in the encoder-cache (EC) store, the proxy forwards the request to PD. PD loads the embeddings through an EC connector and uses them with the text prompt to generate a response.

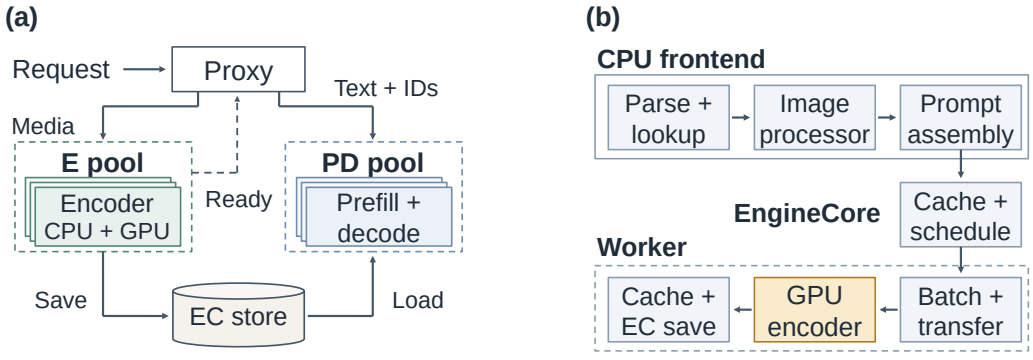


Figure 1: The serving path studied in this paper. (a) Separate E and PD pools exchange encoder embeddings through an EC store; the dashed arrow marks encoder completion. (b) The cache-miss path inside one E instance. Arrows show processing order, not execution time; different requests can overlap across resources.

Inside an E instance. Figure 1(b) shows the image path on a cache miss. The CPU frontend parses the media, builds cache identifiers, and checks the processor cache. For uncached images, the HuggingFace processor resizes and normalizes pixels and prepares the image patches. vLLM assembles the inputs and updates the prompt placeholders that mark where image embeddings belong (vLLM Contributors, 2026b). EngineCore checks the encoder cache and schedules the missing work. The worker batches the inputs, transfers tensors to the GPU, and runs the vision encoder. It then caches the outputs and saves them through the EC connector for PD to load (vLLM Contributors, 2026a).

The processor cache holds processed inputs; the encoder cache holds the resulting embeddings. Neither is the decoder's KV cache.

Why the full path matters. An E instance includes CPU processing, scheduling, GPU execution, and result handoff. These operations contribute to request latency, but some can overlap across requests. In our instrumented setup, the GPU interval also lies inside the worker cycle, so adding both would count it twice. Section 4 measures these relationships. The capacity model therefore needs the resource demand that limits completions under load, rather than GPU forward time or isolated request latency alone.

3 Provisioning Theory for the Tandem System

3.1 Model

A node has $N \geq 2$ GPUs, split into N_E E instances and $N_{PD} = N - N_E$ PD instances. Each request visits E and then PD. Its features are $X_i = (V_i, L_{\text{in},i}, L_{\text{out},i}) \sim F$. Arrivals are Poisson with rate $\lambda > 0$. The vectors $(X_i, \sigma_{E,i}, \sigma_{PD,i})$ are i.i.d. across requests and independent of arrival times. Service requirements are nonnegative with positive finite means and satisfy $\mathbb{E}[\sigma_{E,i} \mid X_i] = D_E(V_i)$ and $\mathbb{E}[\sigma_{PD,i} \mid X_i] = D_{PD}(L_{\text{in},i}, L_{\text{out},i})$. Instances are unit-speed, work-conserving FIFO servers with unlimited buffers. Routing is uniform within each pool, independent across requests and stages, and independent of all arrival, feature, and service data. Transfers are instantaneous, lossless, and nonblocking, with no additional capacity limit. Service laws do not change with the allocation or arrival rate.

The two service requirements of a request may be dependent. With means $\bar{D}_E, \bar{D}_{PD}$, the effective rates are

$$\mu_E(F) = \frac{1}{\mathbb{E}_F[D_E(V)]}, \qquad \mu_{PD}(F) = \frac{1}{\mathbb{E}_F[D_{PD}(L_{\text{in}}, L_{\text{out}})]}. \tag{1}$$

These are serial effective rates, not a renewal claim about concurrent vLLM completions. PD arrivals need not be Poisson. Physical rates require calibration; the PD bounds of Nie et al. (2026) also include packing slack. For encoder resource lanes, average each lane's conditional mean demand before taking the bottleneck. Its reciprocal is a necessary load bound, not automatically attainable capacity (Appendix A.2).

3.2 Capacity and the optimal split

Call a system rate-stable if its unfinished-request count is $o(t)$ almost surely. Proposition 3 proves rate stability below $C(n) = \min\{n\mu_E, (N-n)\mu_{PD}\}$ and linear backlog growth above it. This boundary does not guarantee stationary latency. We optimize

$$\max_{n \in \{0,\ldots,N\}} C(n). \tag{2}$$

The formal empty-pool allocations have zero capacity; physical optimizers lie in $\{1, \ldots, N-1\}$.

Proposition 1 (Optimal allocation)**.** *The relaxation of equation 2 to $n \in [0, N]$ is maximized at*

$$n^* = N\frac{\mu_{PD}}{\mu_E + \mu_{PD}} = N\frac{\bar{D}_E}{\bar{D}_E + \bar{D}_{PD}}, \tag{3}$$

$$C^* = N\frac{\mu_E\mu_{PD}}{\mu_E + \mu_{PD}} = \frac{N}{\bar{D}_E + \bar{D}_{PD}}. \tag{4}$$

An integer maximizer is obtained by comparing $C(n)$ at $n \in \{\lfloor n^* \rfloor, \lceil n^* \rceil\} \cap \{1, \ldots, N-1\}$.

The fractional split assigns GPUs in proportion to mean demand. Appendix A.1 gives the rounding loss, strict integer sizing, and an extension to monotone pool capacities. Linear scaling in equation 4 concerns the fractional benchmark, not the integer optimum.

3.3 Robustness of the integer decision

Proposition 2 (Decision stability)**.** *Let $r = \mu_E/\mu_{PD}$. For $1 \leq n \leq N-1$, allocation n is optimal exactly when*

$$r \in I_n := \left[\frac{N-n-1}{n}, \frac{N-n}{n-1}\right] \cap (0, \infty), \tag{5}$$

with upper endpoint $+\infty$ for $n = 1$. The optimizer is unique in the interval's interior; a finite positive shared endpoint ties adjacent splits.

For $N = 8$, $I_2 = [2.5, 6]$ contains $[r_0/1.5, 1.5r_0]$ at $r_0 = 57/14.3$. This tolerance is specific to this ratio, not independent factor-1.5 errors in both rates. It does not preserve absolute capacity. Appendix A.1 gives exact margins, certificates under rate uncertainty, and worst-case planning.

3.4 Sufficient TTFT rate limits

Mean rates alone do not determine latency compliance. For fixed limits $0 < \bar{\lambda}_s \leq \mu_s$, Proposition 4 in Appendix A.3 assumes stationary component-tail bounds uniform over the allocations and their induced arrival processes. Under those conditions, every $0 < \lambda < G(n)$ meets the stated TTFT budget, where

$$G(n) = \min\{n\bar{\lambda}_E, (N-n)\bar{\lambda}_{PD}\}. \tag{6}$$

Propositions 1 and 2 optimize this certified boundary using $\bar{\lambda}_s$ instead of μ_s. A capacity-optimal n also maximizes G exactly when $\bar{\lambda}_E/\bar{\lambda}_{PD} \in I_n$. This is not exact SLO capacity; positive limits and small discounts are not guaranteed. Empirical rate limits alone are planning estimates. TPOT needs a separate bound; the appendix retains all tail assumptions and probability budgets.

3.5 The planning algorithm

Algorithm 1 Provisioning with capacity or certified TTFT rates

Input: GPU budget N, calibrated stage models or rates, request logs, and optional certified limits or rate bounds.

1. Form $\hat{F}$ from logs and estimate rates using equation 1 or validated resource-load models (Appendix A.2).
2. Use $q_s = \mu_s$ for capacity, $q_s = \bar{\lambda}_s$ for certified TTFT, or positive lower rate bounds for worst-case capacity.
3. Set $n_q^* = Nq_{PD}/(q_E + q_{PD})$. Compare $\min\{nq_E, (N-n)q_{PD}\}$ at the feasible floor and ceiling of n_q^*. Keep the incumbent if tied.
4. Keep the target rate strictly below the selected boundary and check any additional SLO. Monitor absolute capacity and the ratio's decision interval.

4 Calibrating the Encoder Service Demand

Algorithm 1 consumes a demand function for each stage. The PD side comes from Nie et al. (2026). The encoder side is harder than timing a vision transformer: an E instance is a full serving pipeline whose CPU-side work is of the same order as its GPU work. Under concurrency it overlaps the two across requests, so what a request costs the server is neither its GPU time nor its wall time.

We therefore proceed in two stages. Section 4.1 decomposes the single-request timeline into named mechanisms, which fixes the functional form of each component in V and establishes which of them nest. Section 4.2 then fits those forms, converts each component into the active resource time it consumes per completed request under sustained backlog, and takes the demand to be the largest of these across resource lanes, since lanes that run in parallel do not add. The two products serve different purposes: the composed single-request form supplies the encoder part of the non-queueing elapsed time in the latency analysis, while the bottleneck form is the service demand that governs capacity.

All calibration measurements run on the complete EPD chain, never on an isolated encoder, so that every overhead a production request pays is present. The independent variable throughout is V, the number of tokens the encoder emits for a request, which in the vision case is the merged visual token count, fixed by the image resolution after the processor's resize logic and summed over the images of a multi-image request. We parametrize by V rather than by raw input size because the encoder's computation is set by the length of the token sequence it processes, whereas the map from raw inputs to tokens is a convention of each model's processor. The campaign was carried out on a consumer-grade node with four NVIDIA RTX 4090 D GPUs serving Qwen2-VL-2B-Instruct; Section 5 then tests the allocation law on a server-class node that differs in GPU architecture, host CPU and model, with the stage rates measured directly there.

4.1 PIPELINE DECOMPOSITION AND TIMELINE CLOSURE

Span-level instrumentation splits a request's encoder-side wall time into named segments: the HTTP and IPC boundaries it crosses, the media decode and processor path, the scheduling and assembly around them, the GPU forward itself, and the response path. Two rules govern how these compose. Segments occupying disjoint stretches of the timeline add; segments that nest do not, and the worker's CPU envelope nests the GPU forward, so adding the two would count the GPU time twice. A strict-serial audit, one request in flight on a single encoder core, measures this nesting directly and fixes the composition rule. Appendix B gives the full taxonomy.

Under the resulting exclusive partition the timeline closes: across the resolution grid, the classified segments account for the measured wall time with a median unexplained gap of zero. Closure is what justifies the regression that follows, since we are then fitting named mechanisms rather than a black box with a hidden remainder. Figure 2 shows the decomposition across the grid.

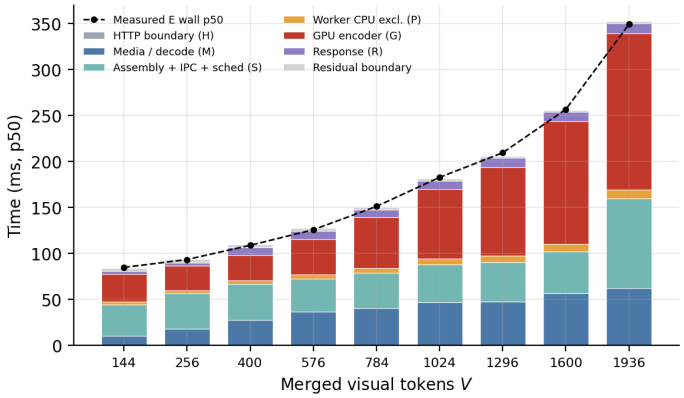


Figure 2: Illustrative strict-serial decomposition of encoder wall time (per-segment medians, 400 requests per point). The GPU lane grows quadratically and dominates at high resolution; CPU-side segments grow linearly. Capacity uses the mean resource demands of Section 4.2, not these medians. Legend S and P correspond to C and $X - G$.

4.2 CALIBRATING THE DEMAND

We consolidate the instrumented path into six resource-aligned components: request ingress (H), media preprocessing (M), EngineCore coordination (C), the worker–GPU cycle (X), GPU execution (G), and result handoff (R). Mechanism fixes the regression form before fitting: control-path terms are constant or linear in V, preprocessing is linear, and the encoder compute terms are quadratic. The GPU interval is nested inside X and is therefore measured separately but never added to it. Appendix C reports all coefficients, R^2, and leave-one-anchor-out errors.

Why fit components separately? A quadratic function can represent the sum of affine and quadratic functions, but this does not make the two fitting procedures equivalent. We choose a form for each component from its mechanism and fit it to that component's measurements. This prevents fluctuations in a component modeled as affine from entering the quadratic coefficient of the composed curve. Separate fitting is therefore not just a different way to write a regression on total time. If all components instead use the same unrestricted quadratic basis, samples, and least-squares weights, the two procedures coincide; Appendix C.1 gives the calculation.

The strict-serial wall time is obtained by summing the mutually exclusive request-path components. Since the GPU interval G is already contained in the worker–GPU cycle X, it is not added separately:

$$\begin{aligned} T_E^{\text{composed}}(V) &= H(V) + M(V) + C(V) + X(V) + R(V) \\ &= 71.35 + 0.07552\,V + 3.072 \times 10^{-5} V^2 \quad \text{ms.} \end{aligned} \tag{7}$$

Equation 7 is a component-wise composition, not a direct fit of total wall time. It describes an isolated request, but not concurrent capacity because different resource lanes overlap across requests.

Let $d_r^{\mathrm{sat}}(V)$ be component r's active time per completion under sustained backlog and k_r its effective parallelism. Define its normalized resource demand as $\widetilde{d}_r(V) := d_r^{\mathrm{sat}}(V)/k_r$. Because lanes overlap, service demand is the largest normalized occupancy rather than their sum:

$$D_E(V) = \max_{r \in \{H,M,C,X,G,R\}} \widetilde{d}_r(V). \tag{8}$$

The worker–GPU cycle is the largest demand throughout the calibrated range, reducing equation 8 to

$$D_E(V) = 23.3416 + 0.0225278\,V + 3.0719 \times 10^{-5} V^2 \quad \text{ms}, \qquad \mu_E(V) = \frac{1000}{D_E(V)}. \tag{9}$$

The shared quadratic coefficient in X and G comes from encoder execution; their difference is CPU-exclusive preparation and completion work around the synchronous GPU call. Control-path, media, and queueing spans affect latency but are either overlapped or load-dependent and do not bind completion rate here. The full component evidence is in Appendix C.

A second encoder. For Qwen2.5-VL-7B-Instruct on the same RTX 4090 D host, the recalibrated component models predict held-out serial E time to 5.25% mean relative error at unseen token counts (Appendix G).

5 Evaluation

We evaluate the full calibration-to-allocation pipeline on four RTX 4090 D GPUs and the provisioning model on eight H800 GPUs. The latter uses independently measured stage rates to test the predicted capacity curve, bottleneck transition, and integer allocation under a different GPU architecture, host CPU, model, workload, and GPU budget.

Both experiments use open-loop Poisson arrivals, SSE streaming, and fixed 128-token outputs. The consumer-grade experiment uses a fixed-V synthetic trace to isolate rate prediction from workload variability, rather than measure a deployment gain. The H800 trace draws from all subjects of MMMU (Yue et al., 2024) validation, preserving the resolution and image-count mix. About 95% of requests have one image, with mean 1.09 and maximum five; no image exceeds $2{,}073{,}600$ pixels. Per-request cache-busting makes every image an encoder miss.

5.1 Baseline comparison: encoder wall-time prediction

We compare our component-wise latency model with the ModServe profiler (Qiu et al., 2025). The released ModServe profiler measures encoder forward time using CUDA events with image tensors already on the GPU. It reports the median of five measurements after two warmup passes, excluding image decoding, preprocessing, and initial image-tensor transfer. A 300-tree random forest predicts latency from parallelism, batch size, and input workload size. We adapt the profiler to Qwen2-VL, using actual visual token count V as the workload feature and fixing parallelism and batch size to one. Both methods use the same encoder, backend, and precision.

We use Qwen2-VL-2B-Instruct on one RTX 4090 D with vLLM 0.22.0, FP16, one CPU thread, and sequential requests with caches disabled. Eight training inputs cover $V \in \{256, 576, 1024, 1600\}$. Training and testing use disjoint source images. Predictions are frozen before testing, and each test input is measured three times after warmup. Errors are computed against each input's median wall time and averaged equally across inputs; signed error is prediction minus measurement.

The fits here are calibrated on this isolated configuration and are not those of Eq. 7, which is measured inside the running service.

Table 1: Prediction errors against full E wall time on 24 input variants from held-out source images. Bias is the mean signed error.

Method	MAE (ms)	RMSE (ms)	MAPE (%)	Bias (ms)
ModServe profiler (Qwen-adapted)	115.03	139.20	67.54	-115.03
Our component-wise fit	16.01	20.71	11.76	$+3.51$

The adapted ModServe profiler underpredicts the CPU-inclusive interval by 115 ms on average, while our fit has a small positive bias and lower absolute error on all 24 inputs (Table 1, Figure 3). The gap

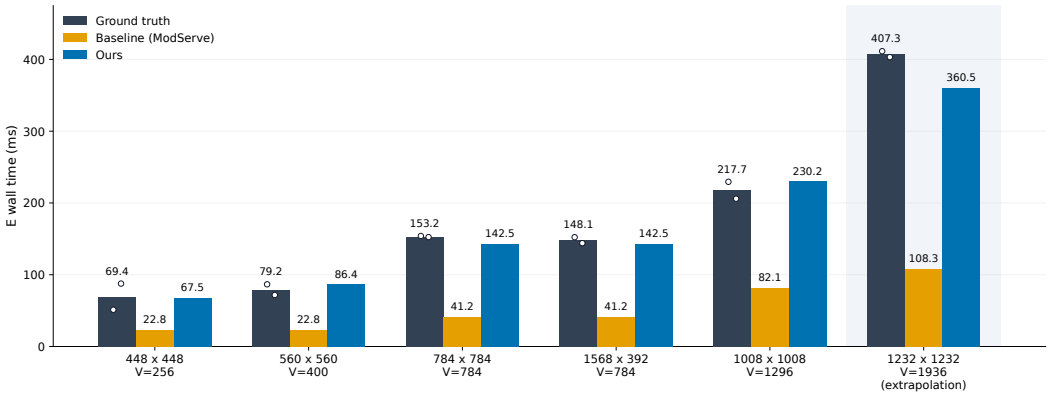


Figure 3: Encoder wall time on six of the 24 input variants, including an aspect-ratio pair at $V = 784$ and one extrapolation point at $V = 1936$.

is one of measured quantity rather than of fitting: the profiler targets an interval that omits part of the work a served request pays for, so its output is not the service demand that capacity planning requires, however accurately it is fitted.

5.2 End-to-end planning on the consumer-grade node

On four RTX 4090 D GPUs serving Qwen2-VL-2B-Instruct, we reuse the encoder calibration of Section 4 and the PD demand model of Nie et al. (2026). Algorithm 1 computes rates from 400 logged requests with $V = 999$, without new rate measurements for allocation selection. Independent bottleneck plateaus at 1:3 and 3:1 validate the E and PD estimates, respectively (Table 2). The encoder prediction is 2.3% above its measured rate and the PD prediction is 21% below it. Both signs follow from how each bound is constructed: the resource-lane bound of Section 3.1 upper-bounds the attainable rate, so the measurement falls below it, while the PD bound carries packing slack and is conservative.

The computed rates give $r = 0.979$ and $n^* = 2.021$, selecting 2:2 with predicted capacity 26.14 requests/s. This split has the highest observed throughput, 23.42 requests/s, which the prediction exceeds by 11.6%. Substituting measured stage rates still predicts 25.56 requests/s, 9.1% above the observation, so most of the gap survives exact rate knowledge and is attributable to the rate ladder and to pool scaling, the latter measured directly on the eight-GPU node in Section 5.4. The 2:2 value is read at the highest pre-collapse rung. All three repeats at 15 requests/s are stable, with a median across repeats of 527 ms for p99 TTFT; higher-rate waves serve only as saturation probes.

Both the computed ratio and the measured ratio $12.78/16.90 \approx 0.76$ lie in the 2:2 decision interval $[0.5, 2]$. The computed ratio has a multiplicative margin of 1.96 to its nearer boundary, so the rate errors leave the selected split unchanged (Proposition 2). Substituting the composed wall time of Eq. 7 for the demand of Eq. 9 moves the ratio to 0.42 and selects 3:1, which measures 16.90 requests/s against 23.42 at 2:2 (Appendix D). The split on this node is therefore decided by the E-side input rather than by the allocation rule. H800 instead selects 2:6 (Section 5.4); both selections maximize observed throughput on their respective nodes. After calibration, target computation requires only logged features, not an allocation sweep. Appendix F costs this contrast on the eight-GPU node of Section 5.3, where a sweep over candidate allocations takes 9 to 19 GPU-hours with instance restarts against under one GPU-hour for the trace calibration. The asymmetry is not a one-off saving: the sweep must be repeated in full at every change of workload, model, or hardware that moves the allocation, whereas the calibration is paid once per hardware–model pair and each later re-planning is arithmetic on logs.

5.3 Setup on the eight-GPU node

We serve Qwen3.5-9B on eight Hopper-class H800 GPUs using vLLM 0.22 with native EPD disaggregation. Encoder-only instances pass embeddings to PD through a shared-storage encoder-cache connector. The 900-request trace is replayed cyclically with cache-busting, producing 3,000 to 12,400 requests per 120 s wave. Throughput is measured over the central 84 s in four 21 s segments.

Table 2: Planning loop on the consumer-grade node. Left: stage rates (requests/s/GPU); measured rates are plateaus at 1:3 (E) and 3:1 (PD). Right: capacity (requests/s); 2:2 is the highest pre-collapse throughput, not a plateau.

Rate	Computed	Measured	Error	Allocation	Predicted	Measured
μ_E	13.07	12.78	2.3%	1:3	13.07	12.78
μ_{PD}	13.35	16.90	21.0%	2:2	26.14	23.42
				3:1	13.35	16.90

Each allocation uses four offered rates at 0.6, 0.85, 1.0, and 1.2 times a planning capacity estimate. For 1:7, the estimate of 48 requests/s yields a top rung of 57.6, which approaches capacity without deep overload.

Client completions, server counters, and cache-store growth agree within 2%, except that encoder counters lead by 3% in the overloaded 4:4 and 5:3 waves, consistent with work accumulating at PD. Below capacity, errors and unfinished requests account for at most 0.12%, except at the 2:6 rung at 85.8 requests/s (Table 5). In four deep-overload waves, 5.8% to 15.9% remain queued at closure. Latency percentiles use completed requests, so compliance is assessed on the sub-capacity waves.

The SLO caps TTFT at 2000 ms and TPOT at 50 ms, each at P90, with per-card concurrency at least six. The evaluation target is a 20% improvement over 4E:4PD. We count errors and unfinished requests as violations. When their share is below 9%, a completed-request p99 within a budget also places the all-request P90 within that budget. Section 5.5 reports compliance on this basis, not a stationary-rate guarantee.

Profiling two GPUs over an afternoon gives $\mu_E \approx 56$ to 57 requests/s, varying with host CPU state, and $\mu_{PD} \approx 14.3$ under deep overload versus 15.2 at light load. A single PD instance meets the TTFT budget at 13.5 requests/s. Using $(\mu_E, \mu_{PD}) = (57, 14.3)$, Proposition 1 gives $n^* = 1.60$ and selects 2:6 with predicted capacity 85.8 requests/s.

5.4 Capacity across allocations

Table 3 and Figure 4 test the capacity curve, levels, and bottlenecks using this single rate pair. Throughput rises from 55.7 requests/s at 1:7 to 86.0 at 2:6, then falls to 74.2, 59.9, and 45.5. The model predicts this ordering and all five values within 6%, with 0.2% error at the selected split. Per-PD throughput declines from 15.15 requests/s with three instances to 14.34 with six, consistent with contention absent from the constant-rate model. Accounting for this pool-size dependence brings agreement to about 1% without changing the selected split.

The observed bottleneck is E at 1:7 and PD at every other tested split, as predicted. At 1:7, offering 57.6 requests/s yields 55.7 completions/s with a growing queue, suggesting capacity between these values and consistent with the independent estimate of 56 to 57. At 2:6, an offered rate of 103 requests/s produces 86.0 completions/s.

Table 3: Measured saturation capacity by allocation, against the model $\min\{N_E\mu_E, N_{PD}\mu_{PD}\}$ with the calibrated pair $(\mu_E, \mu_{PD}) = (57, 14.3)$. †Throughput at the top rung, 57.6 requests/s, where the queue was already building; 1:7 was not driven into deep overload, so its capacity lies between 55.7 and 57.6.

Allocation (E:PD)	Measured cap (req/s)	Model (req/s)	Bottleneck	Per-PD rate (req/s)
1:7	55.7†	57.0	E	—
2:6 (chosen)	**86.0**	85.8	PD	14.34
3:5	74.2	71.5	PD	14.85
4:4 (equal)	59.9	57.2	PD	14.97
5:3	45.5	42.9	PD	15.15

5.5 Throughput under the SLO

We assess whether the capacity-optimal split also maximizes the highest tested SLO-compliant offered rate. Compliance requires both P90 budgets and the concurrency floor, counting errors and unfinished requests as violations. Appendix E reports these waves and their achieved throughput



Figure 4: Eight-GPU allocation sweep. In the throughput panel, dashed levels mark the deep-overload plateau of each allocation driven past capacity (not 1:7).

(Table 5). These finite-wave comparisons complement the sufficient guarantees of Section 3.4; they do not identify exact SLO boundaries.

At 2:6, the highest compliant offered rate is 85.8 requests/s, with 83.8 completions/s and completed-request p99 TTFT and TPOT of 1305 and 36.9 ms. This offered rate is 99.8% of measured capacity. Compliance holds at P90 with incomplete requests counted as violations, since the all-request P90 TTFT lies between the completed-request p90 and p99, 961 and 1305 ms (Table 5). The compliant rungs at 3:5, 4:4, and 5:3 are 3.6%, 4.5%, and 5.7% below capacity. The nonqueueing contribution is estimated at about 150 ms, compared with the 2000 ms budget. Extrapolating the single-PD compliant rate gives $6 \times 13.5 = 81$ requests/s, about 6% below the highest compliant rate observed in this experiment.

Completed-request p99 TPOT ranges from 26.2 to 38.7 ms across all 20 waves, including overload, and does not bind among compliant waves. TTFT rises sharply: at 3:5, 4:4, and 5:3, increasing the offered rate by 9 to 14 requests/s from the last compliant rung to the first overloaded rung increases the completed-request median TTFT seven- to thirteen-fold; unfinished requests remain excluded from these summaries.

The selected 2:6 split also has the highest tested compliant offered rate (Figure 5, in the appendix). The equal 4:4 split meets the SLO at 57.2 requests/s with all requests completed and p99 TTFT 1822 ms; its next rung exceeds its 59.9 requests/s capacity. Against 4:4, the selected split gains 50% in compliant offered rate, 47.8% in achieved throughput, and 43.6% in saturation capacity, each above the 20% target.

6 Conclusions and Limitations

We develop a demand-based framework for dividing a fixed GPU budget between E and PD. Component measurements distinguish encoder resource demand from elapsed latency; an effective tandem model then gives a two-candidate allocation rule and explicit decision intervals. The 4090 experiment evaluates the demand-to-allocation pipeline, while the H800 experiment tests provisioning with independently measured rates. On eight H800 GPUs, the selected split raises measured saturation throughput by 43.6% over an equal split without additional hardware. The computed target complements runtime adaptation.

The capacity and allocation results are exact for the effective-server model; reading them as physical capacity additionally requires mean demands, resource-bound attainability, and pool scaling to hold, and these can change with batching, host contention, or traffic. The TTFT extension is sufficient and conditional rather than an exact SLO-capacity characterization. Experiments cover vision workloads on two platforms, with limited multi-image coverage and no end-to-end recalibration on H800; other modalities, many-image workloads, tighter budgets, and bursty arrivals require further calibration.

AI USE STATEMENT

In preparing this manuscript, we used generative AI tools to provide feedback on methodological framing and experimental presentation, and to assist with manuscript structure, language editing, and LaTeX quality assurance. We did not use generative AI tools to generate data or execute experiments. The authors reviewed all AI-assisted changes, checked the technical content against the underlying measurements and derivations, and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Søren Asmussen. *Applied Probability and Queues*. Springer, 2 edition, 2003.

Shuai Bai et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Hongsen Chen et al. DOPD: A dynamic PD-disaggregation architecture for maximizing goodput in LLM inference serving. *arXiv preprint arXiv:2511.20982*, 2025.

Zhe Chen et al. InternVL: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Yibo Jin, Tao Wang, Huimin Lin, Mingyang Song, Peiyang Li, Yipeng Ma, Yicheng Shan, Zhengfan Yuan, Cailong Li, Yajing Sun, et al. P/D-Serve: Serving disaggregated large language model at scale. *arXiv preprint arXiv:2408.08147*, 2024.

J. F. C. Kingman. The single server queue in heavy traffic. *Mathematical Proceedings of the Cambridge Philosophical Society*, 57(4):902–904, 1961.

Fan Lai et al. TokenScale: Timely and accurate autoscaling for disaggregated LLM serving with token velocity. *arXiv preprint arXiv:2512.03416*, 2025.

Sida Li et al. Taming the chaos: Coordinated autoscaling for heterogeneous and disaggregated LLM inference. *arXiv preprint arXiv:2508.19559*, 2025.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Chengyi Nie, Nian Si, and Zijie Zhou. A queueing-theoretic framework for stability analysis of LLM inference with KV cache memory constraints. *arXiv preprint arXiv:2605.04595*, 2026. Accepted at ICML 2026.

Zhibin Pan et al. RServe: Overlapping encoding and prefill for efficient LMM inference. *arXiv preprint arXiv:2509.24381*, 2025.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. *arXiv preprint arXiv:2407.00079*, 2024.

Haoran Qiu et al. ModServe: Modality- and stage-aware resource disaggregation for scalable multimodal model serving. *arXiv preprint arXiv:2502.00937*, 2025.

Gursimran Singh, Xinglu Wang, Yifan Hu, Timothy Yu, Linzi Xing, Wei Jiang, Zhefeng Wang, Xiaolong Bai, Yi Li, Ying Xiong, Yong Zhang, and Mohammad Salameh. Efficiently serving large multimodal models using EPD disaggregation. *arXiv preprint arXiv:2501.05460*, 2025.

vLLM Contributors. GPUModelRunner: Multimodal encoder execution. vLLM source-code reference, version 0.22.0, 2026a. URL `https://docs.vllm.ai/en/v0.22.0/api/vllm/v1/worker/gpu_model_runner/`. See `_execute_mm_encoder` and `execute_model`. Accessed September 16, 2026.

vLLM Contributors. Multi-modal data processing. vLLM documentation, version 0.22.0, 2026b. URL `https://docs.vllm.ai/en/v0.22.0/design/mm_processing/`. Accessed September 16, 2026.

vLLM Team. Encoder disaggregation for scalable multimodal model serving. vLLM Blog, https://vllm.ai/blog/2025-12-15-vllm-epd, 2025.

Ward Whitt. *Stochastic-Process Limits: An Introduction to Stochastic-Process Limits and Their Application to Queues*. Springer, 2002.

Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. LoongServe: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP)*, 2024.

Jin Xu et al. Qwen2.5-Omni technical report. *arXiv preprint arXiv:2503.20215*, 2025.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9556–9567, 2024. doi: 10.1109/CVPR52733.2024.00913.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

A PROOFS

We distinguish three statements: capacity in the effective-server model, resource-load bounds for the physical encoder, and sufficient latency conditions. The first does not require a renewal arrival process at PD. The third does not infer latency distributions from capacity alone.

Lemma 1 (Saturated serial throughput)**.** *Consider a unit-speed, work-conserving serial instance with permanent backlog and nonnegative service requirements $\sigma_1, \sigma_2, \ldots$. If $m^{-1}\sum_{i=1}^{m}\sigma_i \to d \in (0,\infty)$, its completion rate converges to $1/d$. In particular, the conclusion holds almost surely for i.i.d. requirements of mean d.*

Proof. Set $B_m = \sum_{i=1}^{m}\sigma_i$ and $B_0 = 0$. Because this instance is serial and never idles, B_m is its mth completion time. For $R(t) = \max\{m : B_m \le t\}$, we have $B_{R(t)} \le t < B_{R(t)+1}$. The assumption implies $B_m \to \infty$ and, by inversion, $R(t)/t \to 1/d$. For i.i.d. requirements, the needed average follows from the strong law of large numbers. The serial-service assumption is essential: the completion times of a batched or pipelined engine need not be these partial sums. □

Lemma 2 (Completion rates from sample-path averages)**.** *Consider an initially empty, unit-speed, work-conserving FIFO server. Let a_k be its kth arrival time and $\sigma_k \ge 0$ the service requirement of that arrival. Suppose*

$$\frac{a_k}{k} \longrightarrow \frac{1}{a}, \qquad \frac{1}{k}\sum_{i=1}^{k}\sigma_i \longrightarrow d, \qquad 0 < a, d < \infty.$$

Then its departure rate is $\min\{a, 1/d\}$ and its unfinished-request count divided by time converges to $(a - 1/d)^+$. These are pathwise conclusions; independence of the two sequences is not required.

Proof. Write $B_k = \sum_{i=1}^{k}\sigma_i$ and let b_k be the kth departure time. The FIFO recursion and its expansion are

$$b_k = \max\{a_k, b_{k-1}\} + \sigma_k, \qquad b_k = B_k + \max_{1\le j\le k}\{a_j - B_{j-1}\},$$

where $b_0 = B_0 = 0$. The two assumed limits imply

$$\frac{1}{k}\max_{1\le j\le k}\left|a_j - \frac{j}{a}\right| \to 0, \qquad \frac{1}{k}\max_{0\le j\le k}|B_j - jd| \to 0.$$

To see this, bound the errors for all sufficiently large indices by a fixed small multiple of the index; the finitely many remaining errors, divided by k, vanish. Hence

$$\frac{1}{k}\max_{1\le j\le k}\{a_j - B_{j-1}\} \longrightarrow \left(\frac{1}{a}-d\right)^+, \qquad \frac{b_k}{k} \longrightarrow \max\left\{\frac{1}{a}, d\right\}.$$

Inverting the arrival and departure sequences gives their respective counting rates a and $\min\{a, 1/d\}$. Their difference is the unfinished-request count, which proves the claim. □

Proposition 3 (Tandem throughput)**.** *Start the tandem system empty and fix a split with $N_E, N_{PD} \ge 1$. Let $R(t)$ count requests that have completed both stages, and let $Q(t)$ count all unfinished requests, including those in service. Almost surely,*

$$\frac{R(t)}{t} \longrightarrow \min\{\lambda, N_E\mu_E, N_{PD}\mu_{PD}\}, \qquad \frac{Q(t)}{t} \longrightarrow \big(\lambda - \min\{N_E\mu_E, N_{PD}\mu_{PD}\}\big)^+.$$

Consequently, rates strictly below $C(N_E)$ are rate-stable, while rates strictly above $C(N_E)$ produce linear backlog growth.

Proof. We first analyze E. Independent uniform routing gives each encoder an arrival rate λ/N_E. Its service requirements, in arrival order, are i.i.d. with mean $\bar{D}_E$. Lemma 2 gives the completion rate of every E instance as

$$\min\{\lambda/N_E, \mu_E\} = c_E/N_E, \qquad c_E := \min\{\lambda, N_E\mu_E\}.$$

The next step tracks the PD work carried by these completions. This is needed because the PD arrival stream need not be renewal and its service requirements can be correlated with the work done at E.

It is equivalent to draw each request's independent PD destination at external arrival and reveal it when E finishes. This mark does not affect E service. Enumerate requests at encoder j in their E arrival order. Let $J_{j,k}$ be the PD destination of its kth request, $\sigma_{PD,j,k}$ that request's PD requirement, and $M_j(t)$ its number of E completions by time t. For every encoder j and PD instance h, the strong law gives

$$\frac{1}{m}\sum_{k=1}^{m}\mathbf{1}\{J_{j,k}=h\} \to \frac{1}{N_{PD}}, \qquad \frac{1}{m}\sum_{k=1}^{m}\mathbf{1}\{J_{j,k}=h\}\sigma_{PD,j,k} \to \frac{\bar{D}_{PD}}{N_{PD}}.$$

These are limits along all prefixes, so they still hold at the random indices $m = M_j(t) \to \infty$. No independence between $M_j(t)$ and the service requirements is needed. Since E is FIFO, its completed requests are exactly such prefixes.

Let $A_h(t)$ count arrivals to PD instance h and let $U_h(t)$ be their total PD service requirement. Instantaneous lossless transfer gives

$$A_h(t) = \sum_{j=1}^{N_E}\sum_{k=1}^{M_j(t)}\mathbf{1}\{J_{j,k}=h\}, \qquad U_h(t) = \sum_{j=1}^{N_E}\sum_{k=1}^{M_j(t)}\mathbf{1}\{J_{j,k}=h\}\sigma_{PD,j,k}.$$

It follows that

$$\frac{A_h(t)}{t} \to \frac{c_E}{N_{PD}}, \qquad \frac{U_h(t)}{t} \to \frac{c_E\bar{D}_{PD}}{N_{PD}}.$$

Thus the PD interarrival sequence has inverse counting rate N_{PD}/c_E, and the mean requirement in its own arrival order converges to $\bar{D}_{PD}$. These conclusions also hold with a fixed rule for simultaneous arrivals: the above positive linear limits make individual count and work jumps negligible relative to time. Lemma 2 now gives PD instance h a completion rate $\min\{c_E/N_{PD}, \mu_{PD}\}$. Summing over h proves the claimed rate for $R(t)$. Finally, losslessness gives $Q(t) = A(t) - R(t)$, where $A(t)$ is the external arrival count and $A(t)/t \to \lambda$. □

Meaning of the boundary and routing assumptions. The proposition concerns rate stability, not a finite mean queue length or a stationary latency distribution. Its counting limit also gives $Q(t)/t \to 0$ at $\lambda = C(N_E)$, but this weak conclusion does not justify operating at the boundary. No boundary SLO guarantee is used. The proof extends to any dispatch rule under which each E input has arrival rate λ/N_E and service average $\bar{D}_E$, and each PD input has arrival rate c_E/N_{PD} and service average $\bar{D}_{PD}$. Equal request counts alone are insufficient if routing changes the work mix seen by an instance. Random routing establishes these conditions under our model. A different routing rule must preserve the corresponding count and work limits. Independent random routing preserves Poisson input at E, whereas round-robin does not. Neither routing nor upstream service generally preserves Poisson input at PD. The proof uses the count and work limits instead of a downstream renewal assumption.

A.1 ALLOCATION, ROUNDING, AND RATE UNCERTAINTY

Proof of Proposition 1. The objective in equation 2 is $C(n) = \min\{n\mu_E, (N-n)\mu_{PD}\}$. For real $n \in [0, N]$, the first term increases strictly and the second decreases strictly. They agree at the unique point $n^* = N\mu_{PD}/(\mu_E + \mu_{PD})$. Therefore

$$C(n) = \begin{cases} n\mu_E, & n \le n^*, \\ (N-n)\mu_{PD}, & n \ge n^*. \end{cases}$$

The maximum is attained at n^*, where substitution gives equation 4. Using $\mu_s = 1/\bar{D}_s$ gives the demand forms of equation 3 and equation 4.

Any integer below $\lfloor n^* \rfloor$ is no better than $\lfloor n^* \rfloor$, and any integer above $\lceil n^* \rceil$ is no better than $\lceil n^* \rceil$. This proves the two-candidate rule. Since $0 < n^* < N$, at least one candidate lies in $\{1, \dots, N-1\}$. An endpoint candidate, 0 or N, has zero capacity, while any physical split has positive capacity. Removing endpoint candidates is therefore valid, including when $n^* < 1$ or $n^* > N-1$. □

Exact rounding loss. Let $a = \lfloor n^* \rfloor$ and $b = \lceil n^* \rceil$. With the formal convention $C(0) = C(N) = 0$, the exact ratio is

$$\frac{\max_{n \in \{1,\dots,N-1\}} C(n)}{C^*} = \max\left\{ \frac{\lfloor n^* \rfloor}{n^*}, \frac{N - \lceil n^* \rceil}{N - n^*} \right\}. \tag{10}$$

Indeed, $C(a) = a\mu_E$ and $C(b) = (N-b)\mu_{PD}$, while $C^* = n^*\mu_E = (N-n^*)\mu_{PD}$. Dividing and taking the larger value proves equation 10. It also yields the rate-independent bound

$$\frac{\max_n C(n)}{C^*} \ge 1 - \frac{1}{N}.$$

If n^* is an integer, the ratio is one. Otherwise $b = a+1$; both terms in equation 10 cannot be less than $(N-1)/N$, since that would require simultaneously $n^* > Na/(N-1)$ and $n^* < Na/(N-1)$.

Scaling with the GPU budget. The benchmark equation 4 is linear in N when both stage rates remain fixed. This statement concerns the fractional relaxation. For example, if $\mu_E = \mu_{PD} = 1$, the integer optimum is

$$\max_{n \in \{1,\dots,N-1\}} \min\{n, N-n\} = \lfloor N/2 \rfloor,$$

which need not increase when a single GPU is added. For each new budget, Proposition 1 recomputes the integer optimum. Applying the same model at a different deployment scale also requires that the stage rates and pool scaling remain valid.

Strictly subcritical integer sizing. For a target $\lambda > 0$, the conditions are $N_E > \lambda/\mu_E$ and $N_{PD} > \lambda/\mu_{PD}$. The smallest possible integers are $\lfloor \lambda/\mu_E \rfloor + 1$ and $\lfloor \lambda/\mu_{PD} \rfloor + 1$. Consequently, a target $\lambda > 0$ is strictly subcritical for some physical split exactly when

$$N \ge \left\lfloor \frac{\lambda}{\mu_E} \right\rfloor + \left\lfloor \frac{\lambda}{\mu_{PD}} \right\rfloor + 2. \tag{11}$$

Any additional GPUs can be assigned to either pool without violating the strict inequalities. The fractional quantity $\lambda(\bar{D}_E + \bar{D}_{PD})$ is a sizing benchmark, not a replacement for these integer and strict-load conditions.

Proof of Proposition 2. Divide $C(n)$ by $\mu_{PD} > 0$. The resulting objective is $f(n) = \min\{nr, N - n\}$, so its optimizer depends only on r. By Proposition 1, the two branches are strictly increasing and strictly decreasing, respectively. The integer maximum therefore has at most two adjacent maximizers, and an allocation is optimal exactly when neither available immediate neighbor is better.

For $n < N-1$, the comparison $f(n) \ge f(n+1)$ is equivalent to $nr \ge N - n - 1$. If this last inequality holds, both terms defining $f(n)$ are at least $N-n-1$, whereas $f(n+1) \le N-n-1$. If it fails, $f(n) = nr$ and both terms defining $f(n+1)$ exceed nr. Thus this comparison is equivalent to $r \ge (N-n-1)/n$.

For $n > 1$, the comparison $f(n) \ge f(n-1)$ is equivalent to $(n-1)r \le N-n$. If this inequality holds, $f(n-1) = (n-1)r$ and both terms of $f(n)$ are at least that value. If it fails, $f(n) = N-n$ and both terms of $f(n-1)$ exceed $N-n$. Thus the comparison is equivalent to $r \le (N-n)/(n-1)$.

For $n = 1$, there is no smaller physical allocation and hence no upper restriction; for $n = N-1$, there is no larger physical allocation and the lower restriction is just $r > 0$. Combining the comparisons

gives equation 5. At a finite positive endpoint exactly the corresponding neighboring comparison is tied. Away from those endpoints the maximizing allocation is unique. This also covers $N = 2$, for which 1:1 is the only physical split. □

Multiplicative ratio margins. Write the endpoints of I_n as ℓ_n and u_n, allowing $\ell_n = 0$ or $u_n = +\infty$. Fix a reference ratio $r_0 \in I_n$. For a finite factor $\gamma \geq 1$, every ratio in $[r_0/\gamma, \gamma r_0]$ retains n as an optimizer exactly when

$$\gamma \leq \gamma_{\text{ratio}} := \min\{r_0/\ell_n, u_n/r_0\},$$

where $r_0/0 = +\infty$ and $+\infty/r_0 = +\infty$. Indeed, interval containment is equivalent to $r_0/\gamma \geq \ell_n$ and $\gamma r_0 \leq u_n$. If each stage rate can independently vary by a factor γ around a positive reference pair with ratio r_0, the possible ratios instead fill $[r_0/\gamma^2, \gamma^2 r_0]$. The corresponding condition is $\gamma \leq \sqrt{\gamma_{\text{ratio}}}$. For $N = 8$, $n = 2$, and $r_0 = 57/14.3$, these margins are approximately 1.505 on the ratio and 1.227 on each rate independently. The statement retains optimality, allowing a tie at a decision boundary. Ratios arbitrarily close to such a boundary have largest admissible factors arbitrarily close to one. These decision guarantees do not bound changes in absolute capacity.

A certificate under rate uncertainty. Let the possible rates form the rectangle $\prod_{s\in\{E,PD\}}[\underline{\mu}_s, \overline{\mu}_s]$, with positive finite endpoints. The possible ratios form exactly

$$\left[\frac{\underline{\mu}_E}{\overline{\mu}_{PD}}, \frac{\overline{\mu}_E}{\underline{\mu}_{PD}}\right].$$

The endpoint ratios are attained at opposite corners, and every intermediate ratio is attained by continuity on the rectangle. Proposition 2 therefore makes interval containment both necessary and sufficient for the same n to be optimal throughout this rectangle. Since $C(n)$ is nondecreasing in both rates,

$$\inf_{\mu_s\in[\underline{\mu}_s,\overline{\mu}_s]} C(n) = \min\{n\underline{\mu}_E, (N-n)\underline{\mu}_{PD}\}.$$

Maximizing this expression proves the worst-case planning rule. If the bounds have simultaneous coverage at least $1 - \alpha$, the probability of returning an incorrect interval-containment certificate is at most α. This is an unconditional error statement; it does not assert coverage conditional on a certificate being returned. Constructing valid bounds requires its own statistical assumptions; integrability alone does not provide finite-sample confidence intervals. Planning with point estimates does not itself supply these guarantees.

Nonlinear pool capacities. When per-instance efficiency changes with pool size, separately calibrated pool-capacity functions can replace the linear model:

$$C(n) = \min\{C_E(n), C_{PD}(N-n)\}. \tag{12}$$

Suppose C_E and C_{PD} are nondecreasing on $\{0, \ldots, N\}$, with $C_s(0) = 0$ and $C_s(N) > 0$. A maximizer lies at the two integers bracketing their discrete crossing, with boundary cases included. To prove this, define

$$j = \max\{k \in \{0, \ldots, N\} : C_E(k) \leq C_{PD}(N-k)\}.$$

The set contains zero and not N, so $0 \leq j < N$. For $k \leq j$ the smaller term is $C_E(k)$ and the objective is nondecreasing; for $k \geq j + 1$ it is $C_{PD}(N-k)$ and the objective is nonincreasing. Thus a maximizer is among $j, j+1$. Endpoint candidates can be discarded unless every physical allocation has zero capacity, in which case all are optimal. No continuous interpolation or strict monotonicity is needed. The statement concerns fixed pool-capacity functions; it does not establish that a physical system with interacting pools admits such a representation. Cross-pool contention or transfer limits must also be represented. The constant-rate decision intervals of Proposition 2 do not automatically apply to this model.

A.2 Resource demand for a mixed workload

Resource lanes are capacity constraints, not arbitrary timestamp spans. Let resource ℓ provide $k_\ell > 0$ units of resource time per second. For request i, let $U_{\ell,i} \geq 0$ be its additive requirement on that resource. Assume the requirement vectors are i.i.d. across requests, integrable, and have conditional means $d_\ell(V_i) = \mathbb{E}[U_{\ell,i} \mid V_i]$. The normalized mean demand is $\widetilde{d}_\ell(V) = d_\ell(V)/k_\ell$. Components that consume the same limited resource must be accounted for together. Nested timers for the same

occupation cannot be counted twice. A worker slot and a physical GPU remain distinct constraints even when the GPU interval is nested inside the worker interval.

Lemma 3 (Resource-load bound)**.** *For a finite set of resource lanes with* $0 < \max_\ell \mathbb{E}_F[\widetilde{d}_\ell(V)] < \infty$*, any lossless, rate-stable implementation with arrival counting rate* $\lambda > 0$ *must satisfy*

$$\lambda \le \left(\max_\ell \mathbb{E}_F[\widetilde{d}_\ell(V)] \right)^{-1}. \tag{13}$$

This is a necessary bound, not an assertion that the bound is attained.

Proof. Let $A(t)/t \to \lambda$ be the external counting rate and let $\mathcal{U}(t)$ be the unfinished requests at time t in a system started empty. Rate stability gives $|\mathcal{U}(t)|/t \to 0$. For each resource and each fixed $K > 0$,

$$\frac{1}{t} \sum_{i \in \mathcal{U}(t)} U_{\ell,i} \le K \frac{|\mathcal{U}(t)|}{t} + \frac{1}{t} \sum_{i=1}^{A(t)} U_{\ell,i} \mathbf{1}\{U_{\ell,i} > K\}.$$

The strong law bounds the limit superior by $\lambda \mathbb{E}[U_\ell \mathbf{1}\{U_\ell > K\}]$. Letting $K \to \infty$ makes this zero by integrability. Therefore the completed requests have consumed resource-ℓ work at asymptotic rate $\lambda \mathbb{E}[U_\ell]$; their full requirements differ from those of all arrivals by only $o(t)$. This work cannot exceed $k_\ell t$, so $\lambda \mathbb{E}[U_\ell] \le k_\ell$. Taking the most restrictive lane gives equation 13. □

This calculation explains the order of operations for mixed workloads. Writing the pointwise maximum as $D_E(V) = \max_\ell \widetilde{d}_\ell(V)$ in general gives only

$$\max_\ell \mathbb{E}_F[\widetilde{d}_\ell(V)] \le \mathbb{E}_F \left[\max_\ell \widetilde{d}_\ell(V) \right].$$

For example, equally likely normalized resource vectors $(9, 1)$ and $(1, 9)$ have maximum mean demand 5 but mean pointwise maximum 9. The latter calculation does not represent the same load bound. Its reciprocal is not automatically an achievable rate either.

When one lane ℓ_0 dominates for F-almost every request,

$$\widetilde{d}_{\ell_0}(V) = \max_\ell \widetilde{d}_\ell(V) \quad F\text{-almost surely},$$

the two averages agree and both equal $\mathbb{E}_F[\widetilde{d}_{\ell_0}(V)]$. This is the condition under which the dominant-lane demand in Section 4.2 can be averaged as a single function. The condition must be checked over the support of the workload to be served, not just at one anchor. If the binding lane changes across requests, retain the lane functions and average each one first.

The resource bound equals a sustainable capacity only if the pipeline can attain the limiting resource utilization without an additional scheduling, synchronization, memory, or transport loss. This equality is part of the effective-capacity approximation to be validated. Fitted occupation curves also have estimation error. The curves used in equation 1 must estimate the conditional mean quantities defined above. A curve fitted to conditional medians does not automatically estimate mean resource work; using it in place of a mean-demand curve is an additional approximation that requires validation. Linearity of expectation permits summing exclusive component means, but not their medians. This requirement concerns the calibration targets and cannot be established by the capacity proof alone.

A.3 Latency guarantees and queueing estimates

For a stationary tagged request in the deployed pipeline, use a disjoint partition into nonnegative elapsed-time components

$$\mathrm{TTFT} = W_E + W_{PD} + S. \tag{14}$$

Here W_E, W_{PD} are the assigned queueing portions, and S contains all remaining time through the first emitted token, including encoding, cache transfer, and prefill. It is elapsed time, not $D_E(V)$ or another resource-occupancy demand. Internal waits must be counted once, either in W_s or in S.

Fix a TTFT budget $T > 0$ and violation probability $\epsilon \in (0, 1)$. Choose nonnegative time allowances and error budgets satisfying $\tau_E + \tau_{PD} + \tau_S \le T$ and $\epsilon_E + \epsilon_{PD} + \epsilon_S \le \epsilon$. Write $\mathbb{P}_{n,\lambda}$ for the stationary tagged-request law under the allocation and offered rate in question.

Proposition 4 (Sufficient TTFT rate limits)**.** *Suppose fixed limits $0 < \bar{\lambda}_s \le \mu_s$ have the following property for every $n \in \{1, \dots, N-1\}$ and every $0 < \lambda < \min\{n\bar{\lambda}_E, (N-n)\bar{\lambda}_{PD}\}$: a stationary regime exists and*

$$\mathbb{P}_{n,\lambda}(W_E > \tau_E) \le \epsilon_E, \qquad \mathbb{P}_{n,\lambda}(W_{PD} > \tau_{PD}) \le \epsilon_{PD}, \qquad \mathbb{P}_{n,\lambda}(S > \tau_S) \le \epsilon_S.$$

Then every $0 < \lambda < G(n)$ satisfies $\mathbb{P}_{n,\lambda}(\mathrm{TTFT} > T) \le \epsilon$, where

$$G(n) = \min\{n\bar{\lambda}_E, (N-n)\bar{\lambda}_{PD}\}$$

is the boundary defined in equation 6. Propositions 1 and 2 optimize this certified rate boundary with μ_s replaced by $\bar{\lambda}_s$.

Proof of Proposition 4. Fix n and $0 < \lambda < G(n)$. The hypothesis supplies a stationary law and the three stated probability bounds at this operating point. If all three latency components are within their allowances, then

$$W_E + W_{PD} + S \le \tau_E + \tau_{PD} + \tau_S \le T.$$

Consequently, a union bound gives

$$\mathbb{P}_{n,\lambda}(\mathrm{TTFT} > T) \le \mathbb{P}_{n,\lambda}(W_E > \tau_E) + \mathbb{P}_{n,\lambda}(W_{PD} > \tau_{PD}) + \mathbb{P}_{n,\lambda}(S > \tau_S) \le \epsilon.$$

Let $C^{\mathrm{exact}}_{\mathrm{TTFT}}(n)$ denote the supremum of offered rates admitting a stationary regime with the required TTFT guarantee, with value zero if no positive rate is feasible. The preceding argument proves

$$G(n) \le C^{\mathrm{exact}}_{\mathrm{TTFT}}(n).$$

Equality and feasibility at $G(n)$ itself are not asserted. Because the certified objective has the same minimum-of-two-lines form as $C(n)$, Propositions 1 and 2 optimize G with rates $\bar{\lambda}_E, \bar{\lambda}_{PD}$. This does not show that an optimizer of G maximizes $C^{\mathrm{exact}}_{\mathrm{TTFT}}$; the gap between the two boundaries can depend on the allocation. No independence between the three latency components is needed. □

Allocation under latency discounts. Let $\bar{\lambda}_s = (1-\delta_s)\mu_s$ with $0 \le \delta_s < 1$. The fractional maximizer of the certified TTFT objective is

$$n^*_{\mathrm{SLO}} = N\frac{\bar{\lambda}_{PD}}{\bar{\lambda}_E + \bar{\lambda}_{PD}}. \tag{15}$$

Its integer candidates are the feasible neighbors of n^*_{SLO}, not necessarily those of the saturation maximizer n^*. Proposition 2 gives the exact condition

$$n \in \arg\max_k G(k) \quad \Longleftrightarrow \quad \frac{\bar{\lambda}_E}{\bar{\lambda}_{PD}} = r\frac{1-\delta_E}{1-\delta_{PD}} \in I_n.$$

If the discounts are equal, then $G(n) = (1-\delta)C(n)$ for every allocation. This preserves the full set of integer optimizers and scales both the integer optimum and the fractional benchmark by $1-\delta$. Unequal discounts need not preserve either optimizer. Even if n maximizes G, the conclusion concerns the certified objective, not the unknown exact TTFT-constrained capacity. This limitation also applies when the discounts are equal.

For example, with $N = 8$ and $(\mu_E, \mu_{PD}) = (4, 1)$, the capacity-optimal allocation is $n = 2$, with $C(2) = 6$. If valid certified limits are $(\bar{\lambda}_E, \bar{\lambda}_{PD}) = (1.6, 0.8)$, then

$$G(2) = 3.2, \qquad G(3) = 4,$$

and Proposition 1 applied to these limits selects $n = 3$. Thus even the optimizer of the certified boundary need not coincide with the capacity optimizer.

Uniform limits and allocation-dependent limits. A PD instance receives a routed encoder departure stream. Both its variability and its dependence on service requirements may change with the split. Hence a mean arrival rate alone does not identify the law of W_{PD}. Proposition 4 requires bounds for the actual induced streams throughout its operating region. An isolated PD measurement does not establish this uniform property.

There is a precise way to impose the needed downward-closed rate region without assuming that a tandem delay is monotone. Suppose stationary laws exist for the strictly subcritical operating points under consideration. For $s \in \{E, PD\}$, put $n_E = n$ and $n_{PD} = N - n$ and define

$$\mathcal{A}_s = \left\{ v \in [0, \mu_s] : \begin{array}{l} \mathbb{P}_{n,\lambda}(W_s > \tau_s) \le \epsilon_s \text{ for every } 1 \le n \le N-1, \\ 0 < \lambda < C(n) \text{ with } \lambda/n_s < v \end{array} \right\}.$$

This set contains zero and is downward closed by its definition. If $\bar{\lambda}_s = \sup \mathcal{A}_s > 0$, every $\lambda/n_s < \bar{\lambda}_s$ is covered: there is a $v \in \mathcal{A}_s$ with $\lambda/n_s < v$. One must also bound S over the intersection of the two rate regions. Positive limits are not guaranteed to exist for arbitrary time and error budgets. For example, if $S > T$ almost surely at every operating point, no positive rate can meet the TTFT requirement, even when the capacity model has positive rates. Nor does existence of positive limits imply small discounts. No divergence result is needed, and a limit may equal μ_s. This construction requires genuine bounds over the stated region; a finite collection of compliant measurements does not prove them.

If valid limits are available only separately for each split, write them as $\bar{\lambda}_E(n)$ and $\bar{\lambda}_{PD}(n)$ and check the service and stationarity conditions for that split. The same union-bound proof gives the certified boundary

$$G(n) = \min\{n\bar{\lambda}_E(n), (N-n)\bar{\lambda}_{PD}(n)\}.$$

Without more structure, this objective need not be unimodal, so evaluate all $N-1$ physical allocations. Optimizing the time and error budgets separately for each split likewise need not preserve a single global pair of rate limits.

The role of TPOT. To meet a separate TPOT budget T_{TPOT} at violation level ϵ_{TPOT}, also require $\mathbb{P}_{n,\lambda}(\mathrm{TPOT} > T_{\mathrm{TPOT}}) \le \epsilon_{\mathrm{TPOT}}$ at the selected operating point, or uniformly over the claimed operating region. The TTFT theorem alone does not imply this condition. If the requirement is instead that both budgets hold for the same request with a specified joint probability, split the total error budget across the TTFT and TPOT violations and apply a further union bound.

A rigorous single-server example. For a genuine FIFO $M/G/1$ waiting component with arrival rate ν, i.i.d. service requirement σ independent of arrivals, $d = \mathbb{E}[\sigma] \in (0,\infty)$, and $\mathbb{E}[\sigma^2] < \infty$, the Pollaczek and Khinchine identity is

$$\mathbb{E}[W] = \frac{\nu\mathbb{E}[\sigma^2]}{2(1-\nu d)}, \qquad \nu d < 1 \tag{16}$$

(Asmussen, 2003). For $\tau > 0$ and $\epsilon_w > 0$, Markov's inequality gives

$$\mathbb{P}(W > \tau) \le \frac{\nu\mathbb{E}[\sigma^2]}{2\tau(1-\nu d)} \le \epsilon_w \quad \text{if} \quad \nu \le \frac{2\epsilon_w\tau}{\mathbb{E}[\sigma^2] + 2\epsilon_w\tau d}.$$

Thus

$$\bar{\lambda} = \frac{2\epsilon_w\tau}{\mathbb{E}[\sigma^2] + 2\epsilon_w\tau d} \tag{17}$$

is a conservative, strictly subcritical waiting-time limit. It can be used as one component of Proposition 4 when the waiting component really follows this queue or is stochastically bounded by it throughout the required region. Under the effective-server model, independent random dispatch makes E such a queue at rate λ/N_E. This does not make the real encoder pipeline an $M/G/1$, and it supplies neither the PD nor the remaining-service bound automatically. Mean demand by itself also does not determine $\mathbb{E}[\sigma^2]$.

What single-server monotonicity does establish. For completeness, take a two-sided jointly stationary ergodic sequence $(\sigma_i, a_i)_{i\in\mathbb{Z}}$ with $\sigma_i \ge 0$, $a_i > 0$, $\mathbb{E}[\sigma_i] = d \in (0,\infty)$, and $\mathbb{E}[a_i] = 1$. At arrival rate ν, set the interarrival times to a_i/ν. For $\nu d < 1$, the stationary FIFO waiting time at arrival zero has Loynes' representation

$$W(\nu) = \sup_{k\ge 0} \sum_{i=-k}^{-1} (\sigma_i - a_i/\nu),$$

where the empty sum is zero; see Asmussen (2003). The ergodic theorem makes the summands' partial sums tend to $-\infty$, so the supremum is finite almost surely. Every partial sum is nondecreasing in ν, which gives pathwise monotonicity under this particular common-input coupling. This coupling does not describe arbitrary changes in a tandem's induced PD input when its split changes. It also does not imply divergence at criticality. For the deterministic case $\sigma_i = d$ and $a_i = 1$, $W(\nu) = 0$ for every $\nu < 1/d$.

Optional heavy-traffic approximations. These approximations are not used to prove any capacity, allocation, or SLO guarantee above. For a FIFO $GI/GI/1$ queue with independent i.i.d. interarrival

and service sequences and finite second moments, write C_a^2 and C_s^2 for their squared coefficients of variation and $\rho = \nu d < 1$. The usual heavy-traffic approximation is

$$\mathbb{E}[W] \approx \frac{\rho}{1-\rho} \frac{C_a^2 + C_s^2}{2} d. \tag{18}$$

It is an approximation, not a general upper bound. For fixed-shape interarrival distributions scaled to make ρ tend to one, with fixed service distribution and nonzero limiting variability, the stationary heavy-traffic limit is exponential (Kingman, 1961; Whitt, 2002). This suggests, for fixed $\epsilon_w \in (0, 1)$,

$$w_{1-\epsilon_w}(\rho) \approx \frac{(C_a^2 + C_s^2)d}{2(1-\rho)} \log \frac{1}{\epsilon_w}. \tag{19}$$

Solving this approximation for a queueing allowance τ suggests $\delta \approx (C_a^2 + C_s^2)d \log(1/\epsilon_w)/(2\tau)$ when the resulting $\delta = 1 - \rho$ is small. It is neither a finite-load probability bound nor a guarantee that a given SLO requires only a small discount.

Independent random routing of a Poisson stream gives $C_a^2 = 1$ at an encoder instance. Round-robin to m encoders instead makes an instance's interarrival time the sum of m exponential gaps, with $C_a^2 = 1/m$. At PD, neither a renewal arrival law nor independence between arrivals and service follows from our model. Using equation 18 or equation 19 there is an additional approximation that needs its own validation. Serial dependence can affect delay beyond what a marginal interarrival coefficient of variation captures. None of these approximate tails should be substituted for the uniform guarantees in Proposition 4 without further bounds.

B THE ENCODER TIMELINE

Segments. We use the six-component taxonomy of Section 4.2 throughout; Figure 2 uses a finer segmentation whose S and P correspond to C and $X - G$. Request ingress H contains HTTP receive and body handling; media preprocessing M contains media loading, image decode, and the HuggingFace processor; EngineCore coordination C contains prompt assembly, IPC, enqueue, and scheduling; the worker–GPU cycle X runs from worker entry through preparation, the synchronous GPU call, and postprocessing; GPU execution G is the CUDA-event interval nested inside X; and result handoff R contains result storage and response serialization. Residual boundary gaps are retained only for the closure audit and are not treated as a seventh resource component.

Composition. Since G is nested inside X, the strict-serial exclusive partition adds X once rather than adding X and G:

$$T_{E,\text{serial}}(V) \approx H(V) + M(V) + C(V) + X(V) + R(V) + \text{boundary}. \tag{20}$$

The strict-serial audit confirms the nesting: the measured worker/GPU overlap equals the entire GPU interval at every V, and G occupies 0.26 to 0.52 of the worker-side CPU union. Closure improved from 21 percent unexplained in a pre-study without the boundary taxonomy to a median gap of zero after queue, IPC, HTTP-boundary, and processor-boundary gaps were classified, both in the serial audit and at the concurrent anchor.

The fitted difference between the worker–GPU cycle and physical GPU activity is

$$X(V) - G(V) = 2.3644 + 0.0038246V \quad \text{ms}. \tag{21}$$

It is the CPU-exclusive worker work surrounding the GPU interval: tensor preparation, launch and synchronization handling, and postprocessing. Thus X is the binding worker-slot occupancy, while G separately tests whether physical GPU activity becomes the tighter lane.

Why the GPU lane is quadratic. The encoder runs self-attention over the V merged tokens, so its dominant term costs $\Theta(V^2)$ in both compute and memory traffic, while the position-wise projections and the MLP blocks cost $\Theta(V)$. The quadratic term therefore reflects attention and the linear one everything else inside the tower. Two observations support this reading rather than a fitted coincidence. The GPU lane is the tightest fit of any segment, as Table 4 shows, which is what one expects of a term set by arithmetic rather than by software overhead. And its coefficient survives the passage to saturation almost unchanged, since a GPU lane cannot be overlapped with itself, whereas the linear and constant terms shrink sharply.

Why the CPU-side segments are linear. M, the CPU-exclusive part $X - G$, and R each traverse request data once, at a cost proportional to the number of tokens produced. They carry more residual

scatter than G because encoded byte count, image format, and aspect ratio also enter, none of which is a function of V alone.

Why the control-path terms are small. H and much of C are protocol, IPC, and scheduling costs paid once per request. Their active saturated demands are nearly constant even when their strict-serial spans show a weak linear trend, because most of the latter is overlappable control-path work rather than binding resource occupancy.

C Component Fits

C.1 Component fitting versus fitting the total

Consider two disjoint serial components, with observations c_i and g_i at the same input sizes x_i, and total observations $t_i = c_i + g_i$. The components are fitted by ordinary least squares on their own observations. Suppose the first component is restricted to an affine function and the second to a quadratic, with both forms chosen before fitting. A direct total fit instead regresses t_i on an unrestricted quadratic. The distinction is in the component-specific restrictions and observations, not the degree of the final composed function.

A finite-sample example. Take the synthetic observations

$$(x_1, x_2, x_3) = (0, 1, 2), \qquad (c_1, c_2, c_3) = (0, 1, 0), \qquad (g_1, g_2, g_3) = (0, 1, 4).$$

The affine least-squares fit to the first component has zero slope and intercept $1/3$. The quadratic fit to the second component is exact. Thus

$$\widehat{c}(x) = \frac{1}{3}, \qquad \widehat{g}(x) = x^2, \qquad \widehat{t}_{\mathrm{comp}}(x) = x^2 + \frac{1}{3}.$$

The observed totals are $(0, 2, 4)$, so their quadratic fit is instead $\widehat{t}_{\mathrm{direct}}(x) = 2x$. A quadratic fit to the first component alone would give $2x - x^2$; its negative quadratic coefficient would cancel that of the second component. The affine restriction excludes this cancellation. If the first component's conditional mean is affine, its observed curvature is residual variation rather than curvature of that mean. The example does not itself establish this modeling assumption or an accuracy advantage. Indeed, the direct fit has zero training mean squared error on the totals, whereas the composed fit has error $2/9$.

The role of different regression spaces. For general observations, let B_1 and B_2 have rows $(1, x_i)$ and $(1, x_i, x_i^2)$, respectively, and assume they have full column rank. Write $P_j = B_j(B_j^\top B_j)^{-1}B_j^\top$ for their least-squares projection matrices, and collect the observations into vectors c and g. At the calibration inputs,

$$\widehat{t}_{\mathrm{direct}} = P_2(c + g), \qquad \widehat{t}_{\mathrm{comp}} = P_1 c + P_2 g,$$

so

$$\widehat{t}_{\mathrm{direct}} - \widehat{t}_{\mathrm{comp}} = (P_2 - P_1)c.$$

If the first component's mean vector lies in the column space of B_1, this difference depends only on its residual variation. In contrast, fitting both components with B_2 gives $P_2 c + P_2 g = P_2(c + g)$. Thus separate fitting with a common unrestricted basis and common weights is equivalent to fitting the sum. Different component-specific spaces make that identity inapplicable; they do not guarantee unequal estimates on every dataset or uniformly better prediction.

These calculations concern additive observations, not sums of component percentiles. Conditional mean times over a disjoint partition add, while component quantiles generally do not. In the encoder timeline, the GPU interval is nested inside the worker cycle and must not be added to that cycle again. The example explains estimation of a composed serial-time curve. Capacity planning additionally requires resource demands under load, resource parallelism, and independent validation of the effective-capacity model.

Table 4: Component regressions on the instrumentation testbed, fitted against the form each mechanism predicts; fit and LOOCV values are in milliseconds. *Saturated* gives the normalized demand d_r^{sat}/k_r: X and G keep their strict-serial form, $M(V)/2$ reflects an effective media parallelism of two, and the control-path terms are nearly constant. The slope of H is negligible, -0.07 ms per thousand tokens, so H is treated as constant. C fits poorly because its strict-serial span varies widely at fixed V, but its saturated demand is about 0.35 ms and never binds. X and G share their quadratic coefficient because the quadratic term is the encoder forward, which X contains.

Component	Form	Fit	R^2	LOOCV	Saturated
Request ingress (H)	linear	$0.2899 - 6.65\times10^{-5}V$	.949	.0126	0.2234
Media preprocessing (M)	linear	$14.7019 + 0.02660V$	.919	6.1220	$M(V)/2$
Engine coordination (C)	linear	$28.3797 + 0.02273V$	.538	19.9968	≈ 0.35
Worker–GPU cycle (X)	quadratic	$23.3416 + 0.02253V + 3.072\times10^{-5}V^2$	.995	6.5000	$X(V)$
GPU encoder (G)	quadratic	$20.9772 + 0.01870V + 3.072\times10^{-5}V^2$	.995	6.7382	$G(V)$
Result handoff (R)	linear	$4.6394 + 0.003724V$	.688	1.9349	0.1905
EngineCore input queue	load	excluded from service demand	—	—	—

C.2 Calibration protocol

Algorithm 2 Calibrating $D_E(\cdot)$ on a new hardware–model pair

Input: target node and model; the resolution range covering production traffic.

1. Deploy one E instance with enough PD capacity that PD never binds, with span instrumentation enabled.
2. At five to eight values of V, run a strict-serial pass to fit component forms, then a sustained-backlog pass to measure $d_r^{\mathrm{sat}}(V)$ and lane parallelism k_r.
3. Set $D_E(V) = \max_r d_r^{\mathrm{sat}}(V)/k_r$ and fit the dominant lane to its predicted form.
4. Optionally, validate with two E instances and check that the deep-overload completion plateau agrees with $1/D_E(V)$; this one-time check is priced separately in Appendix F.

Output: $D_E(\cdot)$ for the target pair.

D Wall Time and GPU Time as Allocation Inputs

The allocation depends on the E-side input only through $\mu_E = 1000/D_E$. We hold $\mu_{PD} = 13.35$ requests/s, $N = 4$, and the logged workload at $V = 999$ fixed, and read the E side from the composed strict-serial wall time of Eq. 7 instead of Eq. 9. This gives $D_E = 177.45$ ms and $\hat{\mu_E} = 5.64$ requests/s, a factor of 2.3 below the demand estimate. Wall time charges every overlappable span to the encoder, so the understatement is systematic: adding overlappable time to the demand moves r toward the E-heavy region.

The ratio falls from 0.98 to 0.42, below the lower endpoint of $I_2 = [0.5, 2]$ and into $I_3 = [0, 0.5]$, so the rule selects 3:1 with predicted capacity 13.35 requests/s. That split measures 16.90 requests/s against 23.42 at 2:2 (Table 2), 27.8 percent lower. Accuracy as a latency model and usability as a provisioning input are therefore separate properties.

The GPU interval $G(V)$ of Table 4, the quantity a CUDA-event profiler reports, gives $\hat{\mu_E} = 14.22$ requests/s and $r = 1.07$, which lies in I_2 and selects the same 2:2. Integer robustness admits several inputs at this operating point. The two are separated by level: leaving out the CPU-exclusive worker work overstates capacity by 14.0 percent against 11.6 percent for Eq. 9, and that margin grows at smaller V, where the CPU-side terms carry more of the demand.

E SLO Waves

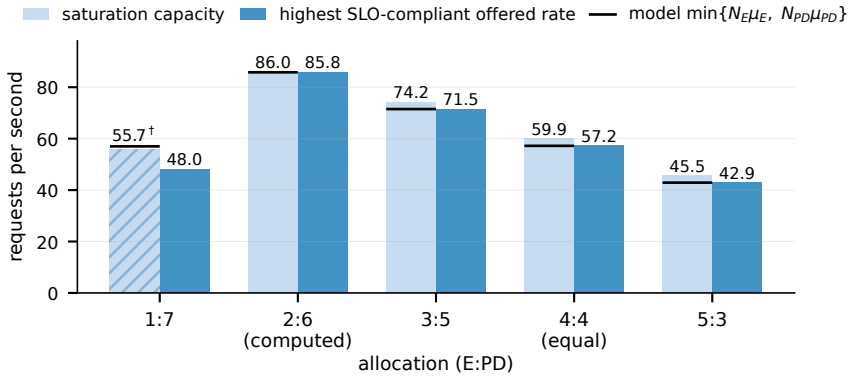


Figure 5: Saturation capacity and the highest SLO-compliant offered rate by allocation, with the model level marked by a black tick; both peak at the computed allocation. †1:7 was not driven into deep overload; 55.7 is the throughput at its top rung and a lower bound on its capacity.

Table 5: Steady waves at and around the SLO point of each allocation, 120 s each, with rates read over the central 84 s. Incompl. is the share of requests that end in an error or remain queued when the wave closes; latency percentiles are over completed requests. Each allocation is swept over a four-rung ladder, so the highest compliant wave listed is the highest rate at which compliance was measured, not the exact boundary. Every wave listed meets both budgets at P90 with incomplete requests counted as violations, and meets the per-card concurrency floor. Bold rows are the rungs compared in Section 5.5. The 2:6 rung at 85.8 is the only listed wave with a substantial incomplete share.

				TTFT (ms)			TPOT	
Allocation	Offered λ	Achieved	Incompl. (%)	p50	p90	p99	p99 (ms)	Conc./card
1:7	48.0	48.5	0.0	355	1560	1895	29.1	23.8
2:6	51.5	51.0	0.0	138	414	882	29.3	22.6
2:6	72.9	74.8	0.0	213	668	1037	34.2	39.8
2:6	**85.8**	**83.8**	**1.9**	**527**	**961**	**1305**	**36.9**	**49.2**
3:5	71.5	69.9	0.0	237	695	1209	37.8	38.8
4:4	48.6	49.2	0.0	138	335	667	33.4	24.0
4:4	**57.2**	**56.7**	**0.0**	**306**	**1014**	**1822**	**38.7**	**33.3**
5:3	42.9	44.1	0.0	521	1293	1807	36.7	27.5

F Cost of Search Versus Calibration

We estimate what it costs to obtain an allocation by search, that is by deploying candidate allocations and measuring them, against what it costs to calibrate once and compute it. The inputs are quantities recorded on the eight-GPU node of Section 5.3; every line item below includes the startup of the instance group it requires.

Unit costs. Bringing up an instance group, E and PD instances together, takes $T_{\text{start}} \in [5, 10]$ minutes regardless of how many GPUs it spans, the spread reflecting host CPU state. A measurement wave is 120 s. A repeated wave, whether to confirm a plateau or to recover from a non-stationary run, may require the group to be brought up again.

Search at deployment scale. Locating a candidate allocation's capacity requires the group on all N GPUs, one startup, and a rate ladder of R waves; the ladder of Section 5.3 has $R = 4$, the minimum that brackets a plateau, and separating candidates whose capacities are close, such as 3:5 and 2:6, needs finer rungs. Per candidate,

$$T_{\text{cand}} \;=\; T_{\text{start}} + R \times 120\,\text{s} \;\in\; [13, 18] \text{ minutes},$$

rising to 18 to 28 minutes when a wave has to be repeated after a restart. The five allocations swept in Section 5.4 therefore cost 65 to 90 minutes of the full node in a single pass and 90 to 140 minutes with one repeat per allocation, scaling by $7/5$ over the seven non-trivial allocations of an eight-GPU node. At eight GPUs held throughout, one search costs 9 to 19 GPU-hours. This is the cost of a single search, and the whole of it is incurred again whenever the workload, model, or hardware moves far

enough that the allocation must be revisited: a sweep produces a number, not a model, so nothing measured in one sweep reduces the cost of the next.

Calibration by rate ladder. Measuring the encoder rate directly at each of K grid resolutions, as the optional deep-overload check of Algorithm 2 does, needs a two-GPU group and, per resolution, R waves; with $K \in [5, 8]$ and $R = 4$ that is 40 to 64 minutes of waves after a single startup, or 45 to 74 minutes of wall clock, and each restart forced by a repeated wave adds another T_{start}. On two GPUs this is 1.5 to 2.5 GPU-hours before repeats.

Calibration by trace. The resource-lane calibration of Algorithm 2 needs the same two-GPU group, one startup, and a single instrumented pass over the trace of about 10 minutes, so $T_{\mathrm{start}} + 10$ minutes, or 15 to 20 minutes of wall clock and 0.5 to 0.7 GPU-hours. It is paid once per hardware-model pair; every later re-planning, including for a changed workload, is arithmetic on logs and costs no GPU time.

Comparison. A single deployment-scale search costs 15 to 30 times the trace calibration before any repetition. The two also scale differently in the number of times the allocation is revisited. Over m such revisions the search side costs m full sweeps, while the calibration side costs one instrumented pass and m recomputations from logs, so the ratio grows without bound in m. The comparison above is therefore the most favourable case for search, namely $m = 1$. Even the ladder calibration on two GPUs, which measures rather than computes the encoder rate, costs three to four times the trace pass and is the more exposed to restarts. The estimate is conservative in two respects: the four-rung ladder is the least a search can do and still locate a capacity, and it charges the search nothing for the SLO waves that a deployment decision would also require.

G Transfer to a Second Encoder

We evaluate whether the component-wise modeling procedure remains applicable to Qwen2.5-VL-7B-Instruct on the same RTX 4090 D host. Component models were calibrated using 60 distinct images at each of five training anchors, $V \in \{64, 256, 576, 1024, 1600\}$. The predicted serial encoder time is

$$\widehat{T}_E^{\mathrm{serial}}(V) = f_H(V) + f_M(V) + f_C(V) + f_X(V) + f_R(V) + f_B(V),$$

where B accounts for boundary intervals outside the five reported request-path components. GPU execution G is nested within X and is not added separately.

All fitted models are frozen before testing. We evaluate three held-out token counts, $V \in \{400, 784, 1296\}$, using 30 distinct new images per point after five warmup requests. Training and test image sets have no SHA-256 overlap, and no test observations are used for refitting. Requests are strictly serial, and the instrumentation verifies per-request timeline closure and $G \subseteq X$. Both calibration and evaluation use the same instrumentation configuration.

Table 6 shows a mean relative error of 5.25% across the three test points, with a maximum of 9.32%. These results support transferring the component-wise modeling procedure to a second encoder through model-specific recalibration, with prediction evaluated at unseen token counts within the calibration range. The validation concerns complete serial E time under the same instrumentation configuration; concurrent capacity and allocation accuracy are separate measurements. Each measured mean and its 95% confidence interval are computed from 30 new test images. Relative error is defined as $|\widehat{T}_E - \overline{T}_E|/\overline{T}_E$, where $\widehat{T}_E$ is the predicted time and $\overline{T}_E$ is the measured mean.

Table 6: Held-out prediction of complete instrumented E wall time for Qwen2.5-VL-7B-Instruct. Times are in milliseconds.

V	Predicted	Measured mean [95% CI]	Rel. error
400	107.44	101.71 [94.94, 108.64]	5.63%
784	128.16	141.33 [136.61, 145.74]	9.32%
1296	166.30	164.98 [156.85, 173.83]	0.80%