RouterWise: Deployment-Aware Prompt Routing for Multi-Model LLMs

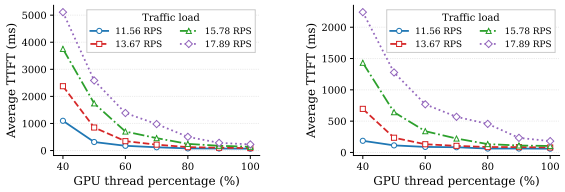
Anonymous ACL submission

Abstract

Multi-model LLM routing aims to direct each prompt to a model whose capability matches its difficulty, balancing quality against cost and latency. Existing routers are trained from prompts alone and treat each candidate model as a black box with fixed per-request latency. In real deployments, however, latency is not a fixed property of a model: it depends on how the model is deployed, its allocated compute and memory, and on the load induced by the routing policy itself. The same model can be the right choice for a prompt under one deployment and the wrong choice under another, and a router calibrated on fixed latencies can produce assignments that are slow, infeasible, or simply suboptimal once deployed. We propose ROUTERWISE, a deployment-aware router that conditions routing decisions on the model's deployment configuration. Given a pool of models, a GPU cluster, and a latency SLO, ROUTERWISE jointly selects a deployment configuration and a routing policy that maximize output quality under the SLO, using a dual-price formulation that reduces score-maximizing routing to per-prompt decisions and setup-specific latency models learned from system profiling. Experiments on three open-source LLMs show that achievable routing quality varies by up to 87% across deployment configurations on the same GPU cluster under the same SLO, demonstrating that the optimal model for a prompt depends not only on the prompt itself, but on the deployment environment.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across reasoning, coding, and multi-step problem solving (Mo et al., 2025; Guo et al., 2025), and are now available in a wide range of sizes from both proprietary providers (OpenAI, 2025; Anthropic, 2025) and the open-source community (Meta AI, 2025; Yang et al., 2025). These models differ substantially in quality, cost, and latency, motivating a central question in LLM serving: how should providers trade output quality against the cost and latency of producing it, given that user prompts themselves vary widely in complexity? Prompt routing addresses this question by dynamically selecting an appropriate model for each input, sending simpler prompts to smaller, faster models and escalating harder queries to larger ones, navigating the quality–cost frontier on a per-prompt basis (Ding et al., 2025; Lu et al., 2024).



(a) 2 GPUs (b) 4 GPUs

Figure 1: Average TTFT of Yi-34B (vLLM, the largest model in our pool) versus GPU thread allocation under varying traffic loads, sharded across 2 and 4 GPUs. Latency varies by over an order of magnitude across configurations at the same load, showing that per-model latency depends jointly on resource allocation and induced load.

Existing routers (Chen et al., 2023; Lu et al., 2024; Ong et al., 2025; Ding et al., 2025, 2024; Chuang et al., 2025; Kasnavieh et al., 2026) are trained from prompts alone and treat each candidate model as if it were a stateless API: a black box with fixed per-request cost and latency, independent of deployment configuration and workload (Ding et al., 2024; Ong et al., 2025). In real deployments, however, models are not stateless APIs. GPU budgets are limited, and when models are co-located on shared GPUs to enable fine-grained allocation of compute and memory (Duan et al., 2024), each model's latency becomes a non-linear function of

its allocated resources and of the request load induced by the routing policy itself (Figure 1). The same model can be the right choice for a prompt under one deployment and the wrong choice under another, and a router optimized against fixed latencies can produce assignments that are slow, infeasible, or simply suboptimal once deployed. Routing and deployment are therefore *tightly coupled*: the resources allocated to a model determine its latency, while the routing policy determines its load and output quality.

In this work, we study routing and resource allocation as a single joint problem. Given a GPU cluster, a pool of models, and a latency service-level objective (SLO), our goal is to determine both (i) the per-model resource allocation, including GPU thread share, memory, and tensor parallelism level, and (ii) a routing policy that assigns prompts to models, so as to maximize output quality subject to the SLO. We propose ROUTERWISE, which searches over feasible deployment setups and, for each, computes the best routing policy under the SLO. The routing subproblem is solved via a *dual-price* formulation that reduces score-maximizing routing under a load budget to simple per-prompt decisions, while setup-specific latency models learned from system profiling capture how each model behaves under different allocations and loads. Our key contributions are: (1) we identify a gap in existing routing methods: they assume fixed per-model latency, which fails under self-hosted deployment; (2) we propose ROUTERWISE, a deployment-aware router that conditions routing decisions on the model's deployment configuration via a dual-price formulation and profiled latency models; (3) we show empirically that achievable routing quality varies by up to 87% across deployment configurations on the same GPU cluster, demonstrating that routing and deployment cannot be optimized independently.

2 Preliminaries

2.1 GPU Sharing Model

LLMs are typically deployed across multiple GPUs using tensor parallelism (TP), which partitions model weights into shards, one per GPU (Xiang et al., 2025). The simplest strategy is to dedicate a full GPU to each shard (Fig. 2(a)), but in multi-model serving this can leave some GPUs under-utilized while others become bottlenecks, since different models face different request loads and

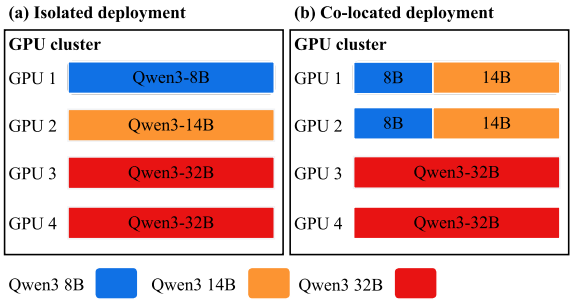


Figure 2: Resource allocation strategies in multi-model LLM serving. Models can be deployed in isolation or co-located on shared GPUs. Co-location enables flexible compute and memory resource sharing and improves utilization.

resource demands. We therefore allow multiple shards to share a GPU, enabling fine-grained allocation of compute and memory across co-located models and shifting resources toward those with higher load.

We enable compute sharing via NVIDIA MPS (NVIDIA, 2025), which schedules CUDA kernels from co-located shards concurrently on the same GPU and lets us cap the fraction of compute resources (active threads) each shard can use. GPU memory is partitioned explicitly across co-located models. Figure 1 shows that increasing the thread cap reduces TTFT under various loads, indicating that latency is highly sensitive to the compute share assigned to each model.

2.2 Routing Model

We assume access to a lightweight router that, for each input prompt x, outputs a score vector

$$s(x) = (s_1(x), \ldots, s_M(x)) \in [0,1]^M,$$

where $s_i(x)$ is the predicted output quality of model i on x. We instantiate the router as a DeBERTa-v3-Large encoder (He et al., 2020) with a multi-head regression layer on the `[CLS]` representation, one head per candidate model, trained on the RouterBench (Hu et al., 2024). The router supplies the per-prompt utility estimates used by our framework, but is not itself the focus of this work; our formulation only assumes access to per-model quality estimates and is agnostic to the router architecture.

Figure 3 shows the predicted score distributions on RouterBench. Larger models tend to score higher on average, but the distributions overlap substantially: a non-trivial fraction of prompts can be served effectively by smaller models. This

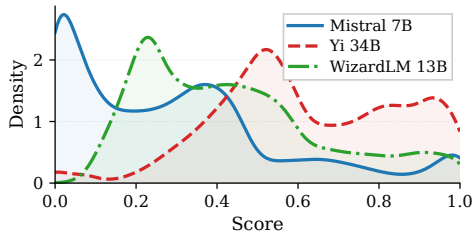


Figure 3: Predicted routing scores on RouterBench. Larger models score higher on average, but the distributions overlap substantially, many prompts can be served effectively by smaller models.

heterogeneity is what makes routing worthwhile—simpler prompts can be handled by smaller models, with larger models reserved for prompts that genuinely benefit from their capability. We use "score" and "output quality" interchangeably throughout: $s_i(x)$ is the router's estimate of model i's output quality on x, calibrated against RouterBench's quality labels.

2.3 System Setup

We consider a multi-model LLM serving system on a GPU cluster with G GPUs and a pool of M models, indexed by $i \in \{1, \ldots, M\}$, serving an incoming workload at global arrival rate λ (requests per second). Each model i is assigned a *setup*

$$\theta_i = (\mathcal{G}_i, \mathrm{tp}_i, \rho_i) \in \Theta_i,$$

where $\mathcal{G}_i \subseteq \{1, \ldots, G\}$ is the set of assigned GPUs, tp_i is the tensor parallelism level, and $\rho_i \in (0, 1]$ is the fraction of GPU compute (thread percentage) allocated to model i. GPU memory for each shard is determined by profiling (see $m_i(\mathrm{tp}_i)$ below). A *system setup* $\Theta = (\theta_1, \ldots, \theta_M) \in \mathcal{S}$ collects the per-model setups, where $\mathcal{S}$ contains all configurations that satisfy deployability constraints: GPU memory capacity, tensor-parallel placement, and the requirement that all shards fit on the available GPUs. For each model i and parallelism level tp_i, we obtain $m_i(\mathrm{tp}_i)$, the minimum memory fraction required by one shard, from profiling; this is used to check shard placement feasibility.

A routing policy $\pi : \mathcal{X} \to \{1, \ldots, M\}$ maps each prompt x, drawn from a workload distribution $\mathcal{D}$, to a deployed model. Our goal is to jointly choose a system setup and a routing policy that maximize output quality subject to a latency target. The two decisions are interdependent: a model's setup determines its latency and throughput under load, while the routing policy determines the load each model receives.

3 Problem Formulation

We formulate the joint resource-allocation-and-routing problem. Let $\Theta \in \mathcal{S}$ denote a feasible system setup. A routing policy makes prompt-level decisions, but over a workload distribution it induces an aggregate traffic split that we capture by a routing fraction vector

$$w \in \Delta_M \triangleq \left\{ w \in \mathbb{R}^M_{\geq 0} \;\middle|\; \sum_{i=1}^{M} w_i = 1 \right\}, \qquad (1)$$

where w_i is the fraction of prompts routed to model i. This workload-level view lets us characterize routing through two quantities: the maximum achievable average score under w, and the resulting latency under setup Θ.

Maximum achievable score. Let $\mathcal{X} = \{x_1, \ldots, x_N\}$ be a finite sample from the workload distribution and $s_i(x_j) \in [0, 1]$ the predicted score of model i on prompt x_j. Given w, the maximum average score achievable subject to model i serving a fraction w_i of the workload is

$$\hat{S}(w) = \frac{1}{N} \max_{\{z_{ji}\}} \sum_{j=1}^{N} \sum_{i=1}^{M} z_{ji}\, s_i(x_j), \qquad (2)$$

subject to

$$z_{ji} \in \{0, 1\}, \quad \sum_{i=1}^{M} z_{ji} = 1 \;\forall j, \quad \sum_{j=1}^{N} z_{ji} = c_i \;\forall i, \qquad (3)$$

where $z_{ji} = 1$ indicates that prompt x_j is assigned to model i, and $c_i \approx N w_i$ is the target count for model i (with $\sum_i c_i = N$).

Latency under setup and routing. Under routing fractions w and global arrival rate λ, model i receives load $\lambda_i = \lambda w_i$. We define

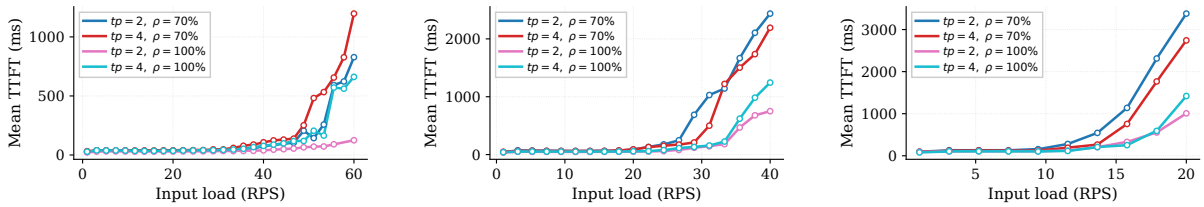
$$L(\Theta, w) = \sum_{i=1}^{M} w_i\, \ell_i(\theta_i, \lambda w_i), \qquad (4)$$

where $\ell_i(\theta_i, \lambda_i)$ is the average latency of model i under its setup θ_i and load λ_i. Depending on the SLO, ℓ_i may denote end-to-end latency, time to first token (TTFT), or time per output token (TPOT).

Joint optimization. The joint problem is

$$\max_{\Theta \in \mathcal{S},\, w \in \Delta_M} \hat{S}(w) \qquad \text{s.t. } L(\Theta, w) \leq \tau, \qquad (5)$$

where τ is the latency target. We focus on average-latency SLOs in this work; extending the formulation to tail-latency targets (P95, P99) is a natural direction for future work.



(a) Mistral 7B (b) WizardLM 13B (c) Yi 34B

Figure 4: Average TTFT versus input load across tensor parallelism levels and GPU compute shares (ρ) for different models in RouterBench obtained by profiling. The latency-load relationship varies substantially across both model size and configuration.

4 RouterWise

This section presents RouterWise in four parts: a dual-price formulation of the score function (§4.1), setup-specific latency models from profiling (§4.2), the optimization procedure that combines them (§4.3), and the deployment of the resulting plan on the cluster (§4.4).

4.1 Dual-Price Formulation of $\hat{S}(w)$

Proposition 1 (Dual-price characterization)**.** *Let $\alpha_i \in \mathbb{R}$ be the dual variable for the count constraint $\sum_j z_{ji} = c_i$ in Eqs. (2)–(3). Dualizing these constraints yields $\min_{\alpha \in \mathbb{R}^M} g(\alpha)$, with*

$$g(\alpha) = \frac{1}{N}\left[\sum_{j=1}^{N} \max_i \big(s_i(x_j) - \alpha_i\big) + \sum_{i=1}^{M} \alpha_i c_i\right]. \tag{6}$$

For any fixed α, the inner maximization decomposes across prompts and is solved by assigning each x_j to

$$i^*(x_j; \alpha) = \arg\max_{i \in \{1,\dots,M\}} \big(s_i(x_j) - \alpha_i\big). \tag{7}$$

Proof. See Appendix A. □

The α_i admit a natural interpretation as model-specific *prices*: increasing α_i makes model i less attractive to all prompts; decreasing it makes the model more attractive. The dual thus reduces the constrained assignment over N prompts to N independent per-prompt argmax decisions governed by these prices.

Proposition 2 (Convexity)**.** *$g(\alpha)$ is convex in $\alpha \in \mathbb{R}^M$.*

Proof. See Appendix B. □

By Proposition 2, g can be minimized via subgradient descent. Let $n_i(\alpha) = \sum_j \mathbf{1}[i^*(x_j; \alpha) = i]$ denote the prompt count assigned to model i under α. A subgradient of g with respect to α_i is $(c_i - n_i(\alpha))/N$, giving the update

$$\alpha_i^{(t+1)} = \alpha_i^{(t)} + \eta_t \frac{n_i(\alpha^{(t)}) - c_i}{N}. \tag{8}$$

Intuitively, over-assigned models have their price raised, and under-assigned models have it lowered.

4.2 Setup-Specific Latency Models

Latency depends not only on input load λ but also on the serving configuration, tensor parallelism (tp) and GPU compute share (ρ), and the dependence is highly non-uniform. Figure 4 shows that the latency-load relationship varies substantially across both configurations and model families: for Mistral 7B, for instance, $\mathrm{tp} = 2$ yields lower TTFT than $\mathrm{tp} = 4$ over a wide range of loads, while the opposite holds for other models. A single unified latency predictor is therefore unlikely to generalize accurately across this space.

RouterWise instead adopts a setup-specific strategy. We discretize the setup space (in our experiments, $\mathrm{tp} \in \{1, 2, 4\}$ and ρ on a discrete grid), profile each (model, setup) pair under multiple input loads, and approximate $\ell_i(\theta_i, \lambda_i)$ as a piecewise-linear interpolation between profiled points. Given a system setup Θ, the overall latency is then

$$L(\Theta, w) = \sum_{i=1}^{M} w_i \, \ell_i(\theta_i, \lambda w_i). \tag{9}$$

We assume ℓ_i depends only on θ_i and λ_i, treating MPS thread caps and explicit memory partitioning as sufficient to mitigate co-location interference. In practice, runtime latency on shared clusters fluctuates due to residual interference, workload burstiness, and infrastructure noise; we therefore treat ℓ_i as a planning model rather than a steady-state guarantee, and RouterWise relies on the relative

Algorithm 1 OPTIMIZEFRACTIONS(Θ, β)

Require: Setup Θ, multiplier β, initial $w^{(0)} \in \Delta_M$, step sizes $\{\eta_t\}$, iterations T
for $t = 0, \ldots, T-1$ **do**
  Solve dual for $\hat{S}(w)$ to obtain $\alpha^\star$
  $g \leftarrow \alpha^\star - \beta \nabla_w L(\Theta, w)$
  $w \leftarrow \Pi_{\Delta_M}(w + \eta_t g)$
end for
return w

Algorithm 2 OPTIMIZEBETA(Θ)

Require: Setup Θ, target τ, bounds $\beta_{\min}, \beta_{\max}$, tolerance ε
$(w^\star, \beta^\star) \leftarrow (\varnothing, \varnothing)$
while $\beta_{\max} - \beta_{\min} > \varepsilon$ **do**
  $\beta \leftarrow (\beta_{\min} + \beta_{\max})/2$
  $w \leftarrow$ OPTIMIZEFRACTIONS(Θ, β)
  if $L(\Theta, w) > \tau$ **then**
    $\beta_{\min} \leftarrow \beta$
  else
    $\beta_{\max} \leftarrow \beta$; $(w^\star, \beta^\star) \leftarrow (w, \beta)$
  end if
end while
return $(w^\star, \beta^\star)$

ranking of setups by latency, which we find is preserved in deployment even when absolute values shift.

4.3 Optimization Procedure

ROUTERWISE solves the joint problem in Eq. (5) through a nested structure: an outer search over feasible system setups, and for each setup an inner score-maximizing routing solver that handles the latency constraint via Lagrangian relaxation (Rush and Collins, 2012). The relaxation introduces a multiplier $\beta \geq 0$ that penalizes latency violations; we bisect on β to find the operating point that meets the latency target τ.

Stage 1: Routing fractions for fixed (Θ, β). For fixed Θ and $\beta \geq 0$, ROUTERWISE solves

$$\max_{w \in \Delta_M} \mathcal{J}(\Theta, w; \beta) \triangleq \hat{S}(w) - \beta\big(L(\Theta, w) - \tau\big) \tag{10}$$

via projected gradient ascent on the simplex. The score gradient is given by the dual prices (Proposition 3 below), and the latency gradient is computed from the slopes of the piecewise-linear curves in Eq. (9):

$$\frac{\partial L(\Theta, w)}{\partial w_i} = \ell_i(\theta_i, \lambda w_i) + \lambda w_i \frac{\partial \ell_i}{\partial \lambda_i}\bigg|_{\lambda_i = \lambda w_i}. \tag{11}$$

Combining the two, $\nabla_w \mathcal{J} = \alpha^\star - \beta \nabla_w L(\Theta, w)$, and the update is $w^{(t+1)} = \Pi_{\Delta_M}\big(w^{(t)} + \eta_t \nabla_w \mathcal{J}\big)$, where Π_{Δ_M} is the simplex projection (Duchi et al., 2008). This is summarized in Algorithm 1.

Proposition 3 (Score gradient via dual prices)**.** *Under the continuous approximation $c_i = N w_i$, if $\hat{S}(w)$ is differentiable at w and $\alpha^\star$ is an optimal dual solution, then $\nabla_w \hat{S}(w) = \alpha^\star$.*

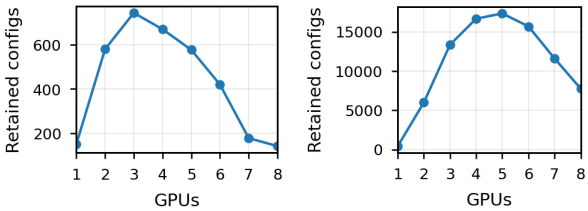
Proof. See Appendix C. □

Stage 2: Bisection over β. Different values of β trace out the score-latency frontier: smaller β favors score, larger β favors lower latency. ROUTERWISE bisects on β to find the operating point that meets the latency target τ (Algorithm 2).

Stage 3: Setup search. We discretize each model's setup over candidate tensor parallelism levels and GPU thread percentages, and form $\mathcal{S}$ as the Cartesian product of these per-model choices. ROUTERWISE enumerates candidates from $\mathcal{S}$, filtering each through two checks. We additionally introduce a utilization threshold $\rho_{\min} \in [0, 1]$ that discards setups leaving the cluster under-utilized. First, the total compute demand $C(\Theta) = \sum_i \mathrm{tp}_i \rho_i$ must satisfy $G\rho_{\min} \leq C(\Theta) \leq G$, discarding setups that under-utilize or over-subscribe the cluster. Second, the induced shards must be placeable on G GPUs under per-shard memory $m_i(\mathrm{tp}_i)$, with no two shards of the same model co-located; we check this via first-fit decreasing. For each surviving setup, ROUTERWISE runs OPTIMIZEBETA and retains the setup with the highest achievable score (Algorithm 3).

4.4 Deployment

Because ROUTERWISE's optimization is driven by profiled latency curves, while runtime latency can fluctuate due to network congestion, neighboring workloads on shared clusters, and other sources of noise, we treat the optimization output as a deployment plan rather than a steady-state guarantee. Given the optimization output $(\Theta^\star, w^\star)$, each model i is launched as a separate vLLM process with tensor parallelism tp_i across its assigned GPUs $\mathcal{G}_i$, with its compute share capped at ρ_i via NVIDIA MPS (`CUDA_MPS_ACTIVE_THREAD_PERCENTAGE`). The router is deployed alongside the model processes: at request time, it computes scores $s(x)$ for the incoming prompt x, the dual-price rule $i^\star(x; \alpha^\star) = \arg\max_i (s_i(x) - \alpha_i^\star)$ selects a model, and the prompt is forwarded to the corresponding vLLM process for generation.



(a) Three-model setting (b) Four-model setting

Figure 5: Number of retained setups as a function of cluster size, for three- and four-model pools. Retained setups are those that pass both the compute-utilization and deployability filters.

Complexity. The cost of ROUTERWISE is dominated by the setup search. Let K_i be the number of discretized configuration choices for model i; naive enumeration of $\mathcal{S}$ scales as $\prod_i K_i$, which grows quickly with M. In practice, the compute-budget and deployability checks prune most candidates: let $\widetilde{\mathcal{S}} \subseteq \mathcal{S}$ denote the retained set. The total cost is $O(|\widetilde{\mathcal{S}}| \cdot T_\beta \cdot C_{\text{OptimizeFractions}})$, where T_β is the number of bisection steps in OPTIMIZEBETA. As shown in Figure 5, $|\widetilde{\mathcal{S}}|$ remains modest in our deployments, peaking around 700 for three models and 17,000 for four models on up to 8 GPUs, making the search tractable even though most prior routing work considers only two-model settings (Ong et al., 2025; Ding et al., 2024; Chen et al., 2023; Ding et al., 2025).

5 Evaluation

We evaluate ROUTERWISE along two complementary axes. **Study 1** (§5.2) asks whether resource allocation matters for routing performance: holding the routing policy fixed at ROUTERWISE's dual-price optimum, we quantify how much achievable quality varies across feasible deployment setups under the same SLO. **Study 2** (§5.3) evaluates ROUTERWISE end-to-end against prior routing systems on a real GPU cluster. Both studies share the experimental setup described next.

5.1 Experimental Setup

Cluster. We evaluate ROUTERWISE on GPU clusters with $G \in \{4, 8\}$ NVIDIA A100 80GB PCIe GPUs, chosen to study the method under both moderate and larger deployment budgets.

Models and dataset. While most prior routing work focuses on routing between two models (Ong et al., 2025; Lu et al., 2024; Ding et al., 2024; Aggarwal et al., 2023; Chen et al., 2023), we consider a more general multi-model setting with three models spanning a range of sizes and capabilities: `mistral-7b-chat` (Mistral AI Team, 2025), `WizardLM-13B-V1.2` (Xu et al., 2023), and `Yi-34B-Chat` (Young et al., 2024). These models are drawn from the RouterBench dataset (Hu et al., 2024), which provides 36,000 prompts paired with model outputs and quality scores derived from benchmark correctness criteria. Each model is served with vLLM (Kwon et al., 2023), which is also used to construct the setup-specific latency profiles.

Algorithm 3 SELECTSETUP

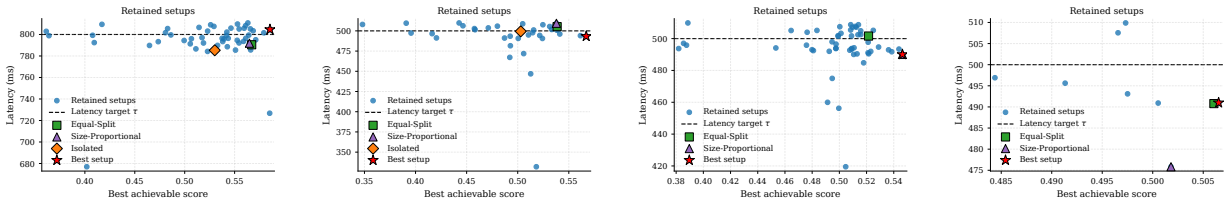
Require: Candidate setup space $\mathcal{S}$, GPUs G, latency target τ, utilization threshold $\rho_{\min}$
Ensure: Best setup-routing pair $(\Theta^\star, w^\star)$

```
(Θ*, w*) ← (∅, ∅);  S* ← −∞
for Θ = (θ_1, ..., θ_M) ∈ S do
    C(Θ) ← Σ_{i=1}^{M} θ_i.tp θ_i.ρ                ▷ compute demand
    if C(Θ) < G ρ_min or C(Θ) > G then
        continue                                   ▷ under-utilized or over-budget
    end if
    Q(Θ) ← []
    for i = 1, ..., M do
        for r = 1, ..., θ_i.tp do
            append m_i(θ_i.tp) to Q(Θ)
        end for
    end for
    if not FFD-FEASIBLE(Q(Θ), G) then
        continue                                   ▷ shards do not fit
    end if
    (w, β) ← OPTIMIZEBETA(Θ)
    if L(Θ, w) ≤ τ and Ŝ(w) > S* then
        S* ← Ŝ(w);  (Θ*, w*) ← (Θ, w)
    end if
end for
return (Θ*, w*)
```

Router. Following §2.2, we train a multi-head DeBERTa-v3-Large (He et al., 2020) router on RouterBench, with one regression head per candidate model. At inference, the router emits a score $s_i(x)$ per model for each input x; these scores feed the dual-price assignment rule $s_i(x) - \alpha_i$.

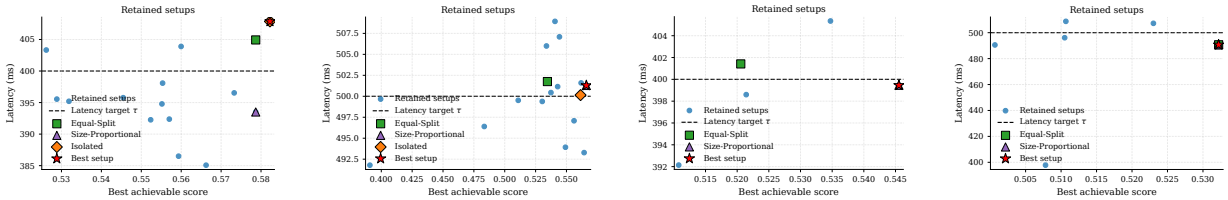
Setup space. We discretize each model's tensor parallelism level over $\{1, 2, 4\}$ and its GPU thread percentage over $\{10, 20, \ldots, 100\}$. Shards of the same model may not be co-located on the same GPU. ROUTERWISE enumerates the Cartesian product of these per-model choices and applies the feasibility checks from §4.3.

Full hyperparameter values are listed in Appendix D.



$\lambda = 50, \tau = 800$ $\lambda = 60, \tau = 500$ $\lambda = 70, \tau = 500$ $\lambda = 80, \tau = 500$

(a) $G = 4$: Maximum achievable score in different setups with 4 GPUs.



$\lambda = 50, \tau = 400$ $\lambda = 60, \tau = 500$ $\lambda = 70, \tau = 400$ $\lambda = 80, \tau = 500$

(b) $G = 8$: Maximum achievable score in different setups with 8 GPUs.

Figure 6: Score-latency scatter plots of retained setups under two settings of G, where retained setups denote deployable setups that also pass the compute-utilization filter. Row (a) shows the case $G = 4$, and row (b) shows the case $G = 8$.

5.2 Study 1: Achievable Performance Across Different Setups

We first study how the deployment configuration alone shapes the quality–latency trade-off available to a router. Holding the routing policy fixed at ROUTERWISE's dual-price optimum, we vary only the deployment setup and measure the achievable score–latency operating point for each. For each (G, λ, τ), we run the routing solver on every retained setup (deployable, passes the compute-utilization filter) and record the achieved score and latency. The resulting points reveal the range of routing quality achievable under the same SLO across different deployments, and thus how much the choice of deployment, independent of the router, constrains what routing can achieve.

Utilization filter. We set $\rho_{\min} = 1$ throughout this study to avoid underutilizing the compute resources. Under NVIDIA MPS, the thread percentage assigned to a process caps the fraction of GPU compute that the corresponding shard can use; if the aggregate cap on a GPU is below 100%, part of its compute cannot be utilized concurrently even when workload is available. Setting $\rho_{\min} = 1$ excludes such trivially under-utilized setups while preventing resource contention, so that any observed score differences reflect *how* compute and memory are distributed across models, not whether compute is left idle.

Resource-allocation baselines. Existing routing methods treat per-model latency as fixed and therefore do not address setup selection, so no prior routing method provides a direct baseline for this study. We instead compare against three allocation strategies, all instantiated on the same discretized setup space as ROUTERWISE:

Equal-Split. Each model is assigned a target compute budget of G/M; we select the feasible setup whose per-model allocations best match this equal-share target.

Size-Proportional. Each model i is assigned a target budget proportional to its parameter count p_i, i.e., $Gp_i/\sum_k p_k$; we again select the closest feasible setup.

Isolated. Co-location is disabled and shards are placed on dedicated GPUs whenever feasible, reflecting a conventional deployment strategy that cannot benefit from fine-grained sharing.

Results. Figure 6 shows the score–latency operating points of all retained setups, with the three baselines and ROUTERWISE's selected setup highlighted. A clear trend emerges across all (G, λ, τ) settings: retained setups span a wide range of achievable scores under the same SLO, with relative scores ranging from 0.30 to 0.56, a spread of up to 87% on the same cluster. This is despite every plotted setup passing the utilization filter, so the variation cannot be attributed to idle GPUs; it

		Score / Average TTFT (ms)		
λ	τ (ms)	ROUTERWISE	FrugalGPT	RouteLLM
50	800	**0.59** / 860	0.59 / 1500	0.58 / 820
60	500	**0.57** / 550	0.57 / 2700	0.57 / 1100
70	500	**0.56** / 565	0.56 / 12300	0.54 / 2500
80	500	**0.51** / 600	0.51 / 24200	0.51 / 7800

Table 1: End-to-end deployment results on 4 A100 GPUs across four offered loads (λ, RPS) under a shared latency target τ.

reflects *how* the available compute and memory are distributed across models. Resource allocation is therefore not a deployment detail but a first-order factor in routing performance, and the three fixed-rule baselines fall short of the setup selected by ROUTERWISE in all settings we evaluate.

Comparing $G = 4$ and $G = 8$, we observe higher achievable scores at $G = 8$ as expected from the larger compute budget, and fewer retained setups: with more resources available, the fraction of discretized configurations that fully saturate the cluster under $\rho_{\min} = 1$ shrinks, leaving a smaller feasible set to search and making the setup search correspondingly faster.

5.3 Study 2: End-to-End Deployment

We evaluate ROUTERWISE end-to-end against two representative routing baselines on a real GPU cluster.

Baselines. We compare against two widely cited routers covering the two dominant routing paradigms: **FrugalGPT** (Chen et al., 2023), which cascades prompts through models from cheapest to most expensive and escalates only when a learned quality score is insufficient, and **RouteLLM** (Ong et al., 2025), which performs pre-generation routing by predicting the best model per prompt. Neither addresses deployment configuration; we therefore deploy each baseline in the conventional manner with models isolated on dedicated GPUs, while ROUTERWISE uses the configuration chosen by its joint optimization. To isolate the deployment contribution from the routing layer, we tune each baseline's quality–cost parameter to match ROUTERWISE's score (within 0.01) at each (λ, τ); the comparison thus reduces to TTFT at equivalent routing quality.

Setup. We deploy on $G = 4$ A100 80GB GPUs and sweep offered load $\lambda \in \{50, 60, 70, 80\}$ RPS under latency targets $\tau \in \{500, 800\}$ ms, reporting mean score and measured average TTFT.

Results. Table 1 shows the outcomes. Because scores are matched by construction, the meaningful comparison is in TTFT. ROUTERWISE stays close to the SLO across all operating points, meeting the target at $\lambda \in \{50, 60, 70\}$ and overshooting by only 100 ms at $\lambda = 80$. Both baselines overshoot far more, and the gap widens sharply with load.

Analysis. The two baselines fail for different but related reasons. FrugalGPT's cascade structure sends every prompt to the cheapest model first and escalates a large fraction to larger models, inverting the natural relationship between model size and serving capacity; since the deployment is isolated and cannot adapt to this skewed load, the cheapest model's queue saturates almost immediately. RouteLLM avoids this cascade overhead by dispatching each prompt directly to a single model, but isolated deployment still leaves the most-used model as a bottleneck. In both cases, the routing decisions themselves are of comparable quality to ROUTERWISE, while the latency gap is mainly attributable to deployment. This isolates the central claim of our paper: when routing quality is held fixed, deployment-aware allocation is what delivers SLO compliance.

6 Conclusion

We have argued that prompt routing cannot be studied in isolation from deployment: when models share a GPU cluster, per-model latency is not a fixed property but a joint function of resource allocation and routing-induced load. Existing routers, trained from prompts alone, miss this dependence and produce assignments that are slow or infeasible under realistic deployment. We proposed ROUTERWISE, a deployment-aware router that jointly selects a deployment configuration and a routing policy via a dual-price formulation over profiled latency models. Experiments on three open-source LLMs show that achievable routing quality varies by up to 87% across deployment configurations on the same cluster. The optimal model for a prompt depends not only on the prompt itself, but also on the deployment environment.

7 Acknowledgements

We used OpenAI ChatGPT to assist with proofreading and improving the clarity and presentation of the manuscript.

8 Limitations

The primary objective of this work is to demonstrate that resource allocation and prompt routing are tightly coupled, and to propose a formal framework for finding an optimal joint solution under a latency SLO. Establishing this coupling, rather than building a production-ready serving system, motivates several assumptions that real deployments will not satisfy.

First, we assume the prompt distribution is fixed and known: the router is trained on a representative sample, and the dual-price optimization is computed against that sample. In practice, prompt distributions shift over time due to workload evolution, domain changes, and user behavior. Adapting routing and allocation decisions online as the distribution drifts is an open question.

Second, our latency models are constructed offline from profiling and treat each model in isolation. Runtime latency on shared clusters fluctuates due to co-location interference, workload burstiness, and infrastructure noise, and we observe modest gaps between profiled and deployed TTFT in our experiments (§5.3). Although ROUTERWISE relies on the relative ranking of setups rather than absolute predictions, incorporating online calibration would tighten the planning loop and improve robustness.

Third, we optimize against a fixed input load λ. Real serving systems face fluctuating arrival rates, and a setup selected for one load may be suboptimal under another. Extending the formulation to multiple load regimes, or to adaptive resource reallocation as load changes, is a natural direction.

The assumptions above scope the contribution of this work without limiting its central claim. Building robust serving systems that operate well under distribution shift, latency noise, and load variation is a substantial research agenda in its own right; the framework presented here provides a principled foundation for that work, and the central observation, that the optimal model for a prompt depends on the deployment environment, holds regardless of which extensions are layered on top.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, and 1 others. 2023. AutoMix: Automatically mixing language models. *Advances in Neural Information Processing Systems*.

Anthropic. 2025. Claude. https://www.anthropic.com.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*.

Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. 2025. Learning to route LLMs with confidence tokens. In *Proceedings of the International Conference on Machine Learning*.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid LLM: Cost-efficient and quality-aware query routing. In *Proceedings of the International Conference on Learning Representations*.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. 2025. BEST-Route: Adaptive LLM routing with test-time optimal compute. In *Proceedings of the International Conference on Machine Learning*.

Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. 2024. Muxserve: flexible spatial-temporal multiplexing for multiple llm serving. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 272–279, New York, NY, USA. Association for Computing Machinery.

Tianyu Guo, Xianwei Zhang, Jiangsu Du, Zhiguang Chen, Nong Xiao, and Yutong Lu. 2025. gllm: Global balanced pipeline parallelism systems for distributed llms serving with token throttling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1725–1741.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-enhanced BERT with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. Routerbench: A benchmark for multi-LLM routing system. In *Agentic Markets Workshop at ICML 2024*.

Hossein Hosseini Kasnavieh, Gholamreza Haffari, Chris Leckie, and Adel N Toosi. 2026. Introlm: Introspective language models via prefilling-time self-evaluation. *arXiv preprint arXiv:2601.03511*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1964–1974.

Meta AI. 2025. The Llama 4 Herd: The beginning of a new era of natively multimodal AI innovation. *Meta AI Blog*. Technical Release and Model Documentation.

Mistral AI Team. 2025. Introducing Mistral 3. Mistral AI Blog. Official Model Release and Technical Documentation.

Zizhao Mo, Jianxiong Liao, Huanle Xu, Zhi Zhou, and Chengzhong Xu. 2025. Hetis: Serving llms in heterogeneous gpu clusters with fine-grained and dynamic parallelism. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1710–1724.

NVIDIA. 2025. *NVIDIA CUDA Multi-Process Service*. NVIDIA Deployment Guide, Release r590.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to route LLMs from preference data. In *Proceedings of the International Conference on Learning Representations*.

OpenAI. 2025. ChatGPT. `https://openai.com`.

Alexander M. Rush and Michael Collins. 2012. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *J. Artif. Int. Res.*, 45(1):305–362.

Yuxing Xiang, Xue Li, Kun Qian, Yufan Yang, Diwen Zhu, Wenyuan Yu, Ennan Zhai, Xuanzhe Liu, Xin Jin, and Jingren Zhou. 2025. Aegaeon: Effective gpu pooling for concurrent llm serving on the market. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pages 1030–1045.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pengcheng Zhao, Chong Feng, and Dacheng Tao. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Alex Young, Bei Chen, Biao Li, Chong Cai, Daya Cao, Guodong Ge, Heng Li, Hongwei Lin, Nanyun Ning, Jiahao Qu, Sivakumar Rajasekaran, and 1 others. 2024. Yi: Open foundation models by 01.ai. *arXiv preprint arXiv:2403.04652*.

A Proof of Proposition 1

In this appendix, we derive the dual-price formulation of the primal score-maximization problem.

Recall the primal problem:

$$\hat{S}(w) = \frac{1}{N} \max_{\{z_{ji}\}} \sum_{j=1}^{N} \sum_{i=1}^{M} z_{ji}\, s_i(x_j) \tag{12}$$

subject to

$$z_{ji} \in \{0,1\}, \qquad \forall j, i, \tag{13}$$

$$\sum_{i=1}^{M} z_{ji} = 1, \qquad \forall j, \tag{14}$$

$$\sum_{j=1}^{N} z_{ji} = c_i, \qquad \forall i. \tag{15}$$

Here, $z_{ji} = 1$ indicates that prompt x_j is assigned to model i, and c_i is the target number of prompts assigned to model i.

We derive the dual by dualizing only the target-count constraints (15). Let $\alpha_i \in \mathbb{R}$ denote the dual variable associated with the constraint

$$\sum_{j=1}^{N} z_{ji} = c_i.$$

The corresponding Lagrangian is

$$\mathcal{J}(Z, \alpha) = \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{M} z_{ji}\, s_i(x_j) + \frac{1}{N} \sum_{i=1}^{M} \alpha_i \left(c_i - \sum_{j=1}^{N} z_{ji} \right). \tag{16}$$

Rearranging (16) gives

$$\mathcal{J}(Z, \alpha) = \frac{1}{N} \left[\sum_{j=1}^{N} \sum_{i=1}^{M} z_{ji} \big(s_i(x_j) - \alpha_i \big) + \sum_{i=1}^{M} \alpha_i c_i \right]. \tag{17}$$

For fixed α, the term $\sum_{i=1}^{M} \alpha_i c_i$ is constant with respect to the assignment variables. Therefore, maximizing $\mathcal{J}(Z, \alpha)$ over Z reduces to maximizing

$$\sum_{j=1}^{N} \sum_{i=1}^{M} z_{ji} \big(s_i(x_j) - \alpha_i\big)$$

subject to the remaining constraints

$$z_{ji} \in \{0, 1\}, \qquad \sum_{i=1}^{M} z_{ji} = 1 \quad \forall j.$$

These constraints couple variables only within each prompt j. Hence, the maximization decomposes across prompts. For a fixed prompt x_j, assigning it to model i contributes

$$s_i(x_j) - \alpha_i$$

to the objective. Since each prompt must be assigned to exactly one model, the optimal choice is

$$i^*(x_j; \alpha) = \arg \max_{i \in \{1,\dots,M\}} \big(s_i(x_j) - \alpha_i\big). \quad (18)$$

Substituting this optimal prompt-wise assignment into (17) yields the dual function

$$\begin{aligned} g(\alpha) &= \max_{Z} \mathcal{J}(Z, \alpha) \\ &= \frac{1}{N} \Bigg[\sum_{j=1}^{N} \max_{i \in \{1,\dots,M\}} \big(s_i(x_j) - \alpha_i\big) \\ &\quad + \sum_{i=1}^{M} \alpha_i c_i \Bigg]. \end{aligned} \quad (19)$$

Therefore, the dual problem is

$$\min_{\alpha \in \mathbb{R}^M} g(\alpha), \quad (20)$$

where $g(\alpha)$ is given by (19).

This proves the dual-price characterization in Proposition 1. In addition, (18) shows that, for fixed dual prices α, the maximizing inner assignment is obtained independently for each prompt by selecting the model with the largest adjusted score $s_i(x_j) - \alpha_i$.

B Proof of Proposition 2

In this appendix, we prove that the dual objective

$$g(\alpha) = \frac{1}{N} \left[\sum_{j=1}^{N} \max_{i \in \{1,\dots,M\}} \big(s_i(x_j) - \alpha_i\big) + \sum_{i=1}^{M} \alpha_i c_i \right]$$

is convex in $\alpha \in \mathbb{R}^M$.

For each prompt x_j, define

$$h_j(\alpha) = \max_{i \in \{1,\dots,M\}} \big(s_i(x_j) - \alpha_i\big).$$

Then $g(\alpha)$ can be written as

$$g(\alpha) = \frac{1}{N} \left[\sum_{j=1}^{N} h_j(\alpha) + \sum_{i=1}^{M} \alpha_i c_i \right].$$

We first show that each function $h_j(\alpha)$ is convex. For any fixed model i, the function

$$h_{j,i}(\alpha) = s_i(x_j) - \alpha_i$$

is affine in α, since it is a constant $s_i(x_j)$ plus a linear term $-\alpha_i$. The function $h_j(\alpha)$ is the pointwise maximum of the finite collection of affine functions $\{h_{j,i}(\alpha)\}_{i=1}^{M}$. A pointwise maximum of affine functions is convex. Therefore, each $h_j(\alpha)$ is convex.

Next, the term

$$\sum_{i=1}^{M} \alpha_i c_i$$

is linear in α, and every linear function is both convex and concave. Therefore, this term is convex.

Finally, $g(\alpha)$ is obtained by summing the convex functions $\{h_j(\alpha)\}_{j=1}^{N}$ and the linear term $\sum_{i=1}^{M} \alpha_i c_i$, and then scaling the result by the positive constant $1/N$. Since nonnegative weighted sums of convex functions remain convex, it follows that $g(\alpha)$ is convex in α.

This proves Proposition 2.

C Proof of Proposition 3

Let

$$V(c) = \frac{1}{N} \max_{\{z_{ji}\}} \sum_{j=1}^{N} \sum_{i=1}^{M} z_{ji}\, s_i(x_j)$$

denote the optimal value of the count-constrained score-maximization problem as a function of the target count vector $c = (c_1, \dots, c_M)$. By Proposition 1, $V(c)$ admits the dual representation

$$V(c) = \min_{\alpha \in \mathbb{R}^M} g(\alpha; c),$$

where

$$g(\alpha; c) = \frac{1}{N} \left[\sum_{j=1}^{N} \max_{i} \big(s_i(x_j) - \alpha_i\big) + \sum_{i=1}^{M} \alpha_i c_i \right].$$

Assume that $V(c)$ is differentiable at the current c, and let $\alpha^\star(c)$ be an optimal dual solution. Since the dependence of $g(\alpha; c)$ on c appears only through the linear term

$$\frac{1}{N}\sum_{i=1}^{M}\alpha_i c_i,$$

the envelope theorem gives

$$\frac{\partial V(c)}{\partial c_i} = \frac{\partial g(\alpha; c)}{\partial c_i}\Big|_{\alpha=\alpha^\star} = \frac{\alpha_i^\star}{N}.$$

Now define $\hat{S}(w) = V(c)$ under the continuous approximation $c_i = Nw_i$. Applying the chain rule,

$$\frac{\partial \hat{S}(w)}{\partial w_i} = \sum_{k=1}^{M}\frac{\partial V(c)}{\partial c_k}\frac{\partial c_k}{\partial w_i}.$$

Because $c_k = Nw_k$,

$$\frac{\partial c_k}{\partial w_i} = N\,\mathbf{1}[k=i].$$

Therefore,

$$\frac{\partial \hat{S}(w)}{\partial w_i} = \frac{\alpha_i^\star}{N}\cdot N = \alpha_i^\star.$$

Hence,

$$\nabla_w \hat{S}(w) = \alpha^\star.$$

This proves the proposition.

D Hyperparameters

We document the hyperparameter values used in our experiments (§5). All values were chosen on a held-out subset of RouterBench and held fixed across all reported runs.

Setup search space (§4.3). Tensor parallelism levels are drawn from $\{1, 2, 4\}$ and GPU thread percentages from $\{10, 20, \ldots, 100\}$. These values are also used to profile different model latencies.

Feasibility check. The first-fit-decreasing placement uses a memory slack tolerance of 0.02 to absorb minor profiling jitter. The compute-utilization filter requires the aggregate thread sum to reach the cluster's compute budget ($\rho_{\min} = 1$).

Fraction optimizer (Algorithm 1). Projected gradient ascent on the simplex runs for 200 iterations with constant step size $\eta_t = 0.1$ (the step size in the update rule in §4.3), initial Lagrange multiplier $\beta_0 = 0.1$, momentum 0, early-stopping patience 10, and objective tolerance 10^{-6}.

Dual-price subgradient (§4.1). The dual problem is solved with 300 subgradient iterations, initial step size $\eta_0 = 10^{-4}$, count tolerance 1, and a small tie-breaking noise of 10^{-9} added to scores to avoid plateaus.

β bisection (Algorithm 2). The Lagrange multiplier β is searched via bisection over $[0, 5]$ for 80 outer steps with multiplicative step decay 0.99.