COMPAREROUTE: LEARNING RELATIVE EXPERT CAPABILITY FOR PREFILL-BASED LLM ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM routing selects an expert for each query before answer generation. Many routing approaches try to reduce success-probability prediction error. However, this does not necessarily lead to better routing. The relative ordering of experts is key to routing decisions. We propose CompareRoute, a router that learns relative ordering from a frozen LLM Observer's prefill hidden states. It dynamically combines hidden states across Observer layers for each query and expert, then models and predicts relative ordering for each expert. We evaluate CompareRoute on three benchmarks with five heterogeneous experts. In-domain success rates improve by an average of 1.06 percentage points over the strongest evaluated baseline. Including Observer input tokens, total token usage decreases by an average of 23.71% across benchmarks relative to the best-single reference. These results highlight the potential of CompareRoute to improve response quality while reducing token usage in multi-LLM deployments.

1 INTRODUCTION

LLM routing has attracted growing interest as a way to improve response quality while reducing inference costs. Its central idea is to exploit complementary model capabilities by assigning each query to the expert most likely to succeed. Existing approaches learn query-dependent routing policies from preference or performance supervision (Ong et al., 2025; Chen et al., 2024; Feng et al., 2025). Their effectiveness depends on identifying which expert has an advantage on a given query before any candidate answer is available.

Prefill representations offer a source of evidence for this pre-generation decision (Cencerrado et al., 2026; Lugoloobi et al., 2026). An independent *Observer* can thus provide information about a heterogeneous expert pool without requiring an inference pass through every candidate (Figure 1). SharedTrunkNet uses prefill evidence to jointly predict the experts' success probabilities and derives routing decisions from these estimates (Varshney et al., 2026). However, more accurate probability predictions do not necessarily yield better routing decisions. For example, suppose two experts have true success probabilities of 80% and 70%. Predictions of 74% and 75% have a smaller mean absolute error but select the weaker expert. Predictions of 90% and 80% have a larger error but select the better one. The relative ordering of experts determines the routing decision. We therefore need to go beyond absolute probability prediction: explicitly modeling relative ordering between experts is key to improving routing performance.

We propose CompareRoute to learn relative expert capability from the multilayer states of a shared frozen Observer. It first constructs a query-dependent representation for each expert by dynamically weighting the Observer's layer states. These representations then feed a shared antisymmetric comparator that directly predicts pairwise success-rate differences. The comparisons are aggregated into relative scores for expert ranking. When training, our comparison loss supervises all expert pairs, with additional weight assigned to pairs involving the router's current top choice in proportion to their empirical success-rate gaps. At inference, the router selects the best expert. Only the selected expert generates an answer.

We evaluate CompareRoute on SuperGPQA, LiveCodeBench-v6, and MMLU-Pro with five heterogeneous experts. In-domain evaluation shows improvements over the evaluated routing baselines on all three benchmarks, with an average success-rate gain of 1.06 percentage points over the strongest

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.

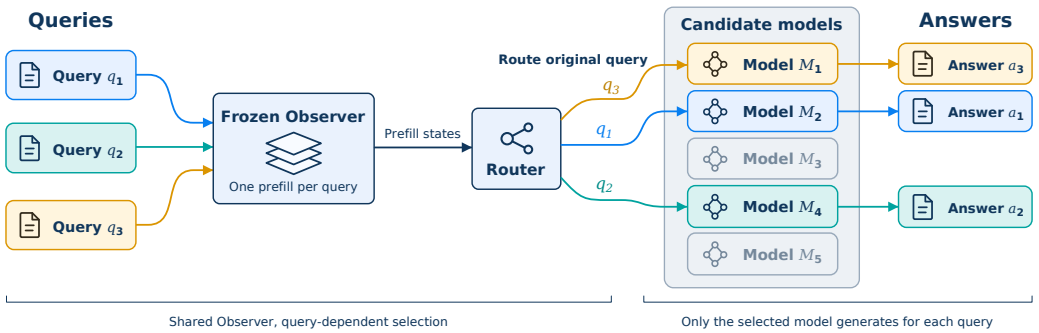

Figure 1: Query-dependent routing with a shared frozen Observer. One prefill per query provides routing evidence; only the selected model generates an answer.

baseline. Including Observer input tokens, total token usage decreases by an average of 23.71% across benchmarks relative to the best-single expert reference.

Our contributions are as follows:

- We propose CompareRoute, a router that explicitly learns query-specific success-rate differences between experts. It predicts pairwise differences and aggregates them into relative scores for expert selection.
- We design expert-conditioned dynamic layer aggregation to extract routing evidence from a shared frozen Observer. The layer weights adapt to both the query and the expert.
- We develop a comparison loss that supervises all expert pairs and gives extra weight to those involving the router's current top choice. This extra weight is proportional to each pair's empirical success-rate gap.
- We demonstrate in-domain improvements over all evaluated baselines on three benchmarks with five heterogeneous experts. Controlled ablations support the effectiveness of adaptive aggregation, explicit expert comparison, and the comparison weighting scheme.

2 Related Work

Predictive LLM routing. Predictive routing selects an expert before answer generation. It relies on query-dependent estimates of model performance. The goal is often to improve response quality and reduce inference cost. RouteLLM and Hybrid LLM use preference or quality estimates with adjustable thresholds to route between stronger and weaker models (Ong et al., 2025; Ding et al., 2024). For selection across larger model pools, RouterDC and GraphRouter capture query–model relationships through dual contrastive learning and heterogeneous graphs, respectively (Chen et al., 2024; Feng et al., 2025). IRT-Router models interactions between model abilities and query attributes (Song et al., 2025). EquiRouter uses model-conditioned representations and pairwise ranking supervision to mitigate routing collapse (Lai & Ye, 2026). RouterBench provides an evaluation framework for multi-LLM routing, including KNN and MLP baselines that predict model performance from query embeddings (Hu et al., 2024; Li, 2026; Li et al., 2026). Our approach learns query-specific relative expert capability. It explicitly predicts pairwise success-rate differences to guide expert selection.

Pre-generation capability estimation. Pre-generation activations can provide predictive signals of model success (Cencerrado et al., 2026). Lugoloobi et al. (2026) train linear probes on these activations to predict success and guide model routing. Most closely related, Varshney et al. (2026) introduce encoder–target decoupling, using an open-weight encoder's prefill activations to predict other models' correctness. Their SharedTrunkNet jointly estimates correctness probabilities for candidate models from concatenated prefill features. Building on this setting, we dynamically aggregate multiple layers of a shared frozen Observer, with weights conditioned on both the query and expert identity. The resulting expert-specific representations support explicit prediction of pairwise success-rate differences for expert selection.

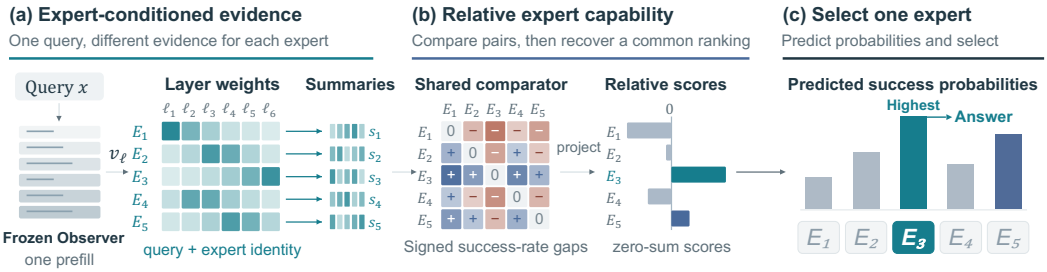

Figure 2: Overview of CompareRoute. (a) Adapts prefill evidence to each query–expert pair. (b) Explicitly models pairwise differences in expert capability. (c) Converts relative scores into success probabilities for routing.

Comparative and decision-focused learning. RouteLLM's matrix-factorization router learns model scores from pairwise preferences to estimate preference win probabilities (Ong et al., 2025). Our comparison target is the empirical success-rate difference estimated from repeated offline samples, expressing relative capability in terms of answer correctness. More broadly, decision-focused learning aligns training with downstream decision quality and can be formulated as learning to rank feasible solutions (Mandi et al., 2022). Causal LLM Routing combines counterfactual utility estimation from observational feedback with differentiable regret surrogates, including softmax-weighted regret (Tsiourvas et al., 2025). Our setting provides offline outcome samples for every query–expert pair, enabling direct supervision of empirical success rates and their differences. LambdaLoss incorporates ranking metrics into loss design and encompasses LambdaRank (Wang et al., 2018). We combine success supervision and continuous regret with a pairwise comparison loss. It supervises all expert pairs, with extra weight on those involving the router's current top choice.

3 Method

As shown in Figure 2, CompareRoute learns capability differences between experts to guide routing selection. A frozen Observer first performs a single prefill pass on the input query. The router dynamically aggregates the Observer's hidden states into a query-dependent summary for each expert. It then compares these summaries to predict pairwise capability differences. Then we route the query to the best expert for the query. Only the selected expert generates an answer.

3.1 Expert-conditioned dynamic layer aggregation

We extract the layer states of the last valid token from the Observer. Then we aggregate these states into representations for assessing expert capability. Using the same fixed aggregation for all experts limits our ability to adjust layer weights for each expert. For example, comparing code and math experts may draw on certain layers. Comparing code and QA experts may emphasize different layers. We therefore learn dynamic layer weights conditioned on the query and expert identity. This produces a representation $s_i(x)$ for each expert, providing evidence for subsequent pairwise comparisons.

To prepare these states for aggregation, we apply PCA and standardization separately to each selected Observer layer. This gives a feature vector $u_\ell(x)$ for each layer $\ell \in \mathcal{L}$. Here, $\mathcal{L}$ contains the L selected layers. These transformations are fitted on the corresponding outer training split and remain fixed during router training and inference.

PCA is fitted independently for each layer. We therefore use separate learnable projections to map the layer features into a common dimension:

$$v_\ell(x) = \mathrm{GELU}\big(W_\ell u_\ell(x) + b_\ell\big) \in \mathbb{R}^d. \tag{1}$$

We next compute query- and expert-dependent weights over these projected layer vectors. Let e_ℓ^{layer} and e_i^{expert} be learnable layer and expert embeddings, respectively. For expert i, a shared scoring network evaluates each layer using its projected state and these embeddings:

$$g_{i\ell}(x) = w_g^\top \tanh\big(W_g v_\ell(x) + e_\ell^{\text{layer}} + e_i^{\text{expert}}\big). \tag{2}$$

We normalize the scores across layers to obtain aggregation weights. We then take a weighted sum of the layer vectors to form the expert-specific summary:

$$\alpha_{i\ell}(x) = \frac{\exp(g_{i\ell}(x)/\tau_L)}{\sum_{k\in\mathcal{L}} \exp(g_{ik}(x)/\tau_L)}, \qquad s_i(x) = \sum_{\ell\in\mathcal{L}} \alpha_{i\ell}(x) v_\ell(x). \tag{3}$$

Here, $\tau_L > 0$ is the layer-attention temperature. All experts share the layer vectors and scoring network. The weights depend on the query through $v_\ell(x)$ and on the expert identity through e_i^{expert}. The expert embeddings are shared across queries.

3.2 From expert comparisons to relative scores

The summaries $s_i(x)$ provide expert-specific evidence; the comparison module turns this evidence into estimates of relative advantage. It first predicts differences between pairs of experts, then reconciles those differences into a common set of scores for routing.

We first compare each expert pair using an ordered representation ξ_{ij} that combines s_i and s_j, their expert identity embeddings, and a symmetric comparison context $c_{ij} = c_{ji}$. Appendix A.1 specifies the context and input construction. A two-layer MLP f_θ, with a GELU hidden activation, is shared across all pairs and both input orders. We define the predicted difference as

$$d_{ij}(x) = \tanh\left(\frac{f_\theta(\xi_{ij}) - f_\theta(\xi_{ji})}{2}\right). \tag{4}$$

This construction guarantees $d_{ji} = -d_{ij}$ and bounds each prediction to $(-1, 1)$. We train $d_{ij}(x)$ to predict $\Delta_{ij}(x)$: its sign predicts the advantaged expert, and its magnitude estimates the success-rate gap.

We then aggregate the $N(N-1)/2$ pairwise predictions into one relative score per expert:

$$r_i(x) = \frac{1}{N}\sum_{j=1}^{N} d_{ij}(x), \qquad d_{ii}(x) = 0. \tag{5}$$

These scores sum to zero and are the unique solution of

$$\min_{\{r_i\}:\,\sum_{i=1}^N r_i = 0} \sum_{1\le i<j\le N} \big[d_{ij} - (r_i - r_j)\big]^2. \tag{6}$$

Thus, the aggregation projects potentially inconsistent pairwise comparisons onto a single set of zero-sum expert scores. If $d_{ij} = \hat{p}_i - \hat{p}_j$, then $r_i = \hat{p}_i - \frac{1}{N}\sum_j \hat{p}_j$: the scores recover the empirical relative advantages and preserve the success-rate ordering.

Although relative scores suffice to rank experts within each router, we also estimate success probabilities to support success-rate supervision. Specifically, we map these scores to probabilities using a common query-dependent offset $b(x)$, predicted from the concatenated summaries $S(x) = [s_1; \ldots; s_N]$:

$$z_i(x) = b(x) + a\, r_i(x), \qquad q_i(x) = \sigma(z_i(x)), \qquad a = \text{softplus}(\gamma) + \varepsilon, \tag{7}$$

where γ is a learnable scalar and $\varepsilon > 0$ is a small constant. Because a is positive and $b(x)$ is common to all experts, the ranking within each router is determined entirely by $r_i(x)$. This is a constrained probability mapping learned through success supervision; it does not assume exact recovery of $p_i(x)$ from the predicted differences. Appendix A.2 gives the detailed architecture.

3.3 Decision-relevant learning and inference

Errors in relative advantage matter for routing when they alter the selected expert. With finite data and model misspecification, reducing average probability prediction error does not guarantee a better choice. Our routing supervision therefore combines three roles: success BCE supervises numerical probability estimates, continuous regret favors higher-success selections, and comparison supervision learns the relative advantages involved in those selections. Figure 3 illustrates their roles.

For each training query x, let $\hat{p}_i(x)$ be the fraction of correct answers among K offline samples from expert E_i. We define the pairwise supervision target as $\Delta_{ij}(x) = \hat{p}_i(x) - \hat{p}_j(x)$.

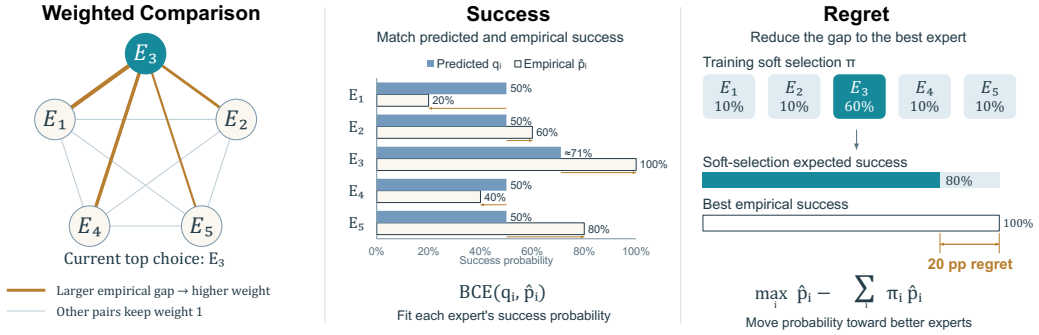

Figure 3: Three training losses. Left: larger empirical gaps give top-choice pairs more weight. Middle: BCE fits predicted to empirical success rates. Right: regret measures the gap between best and expected empirical success.

For a training batch B, we then supervise these probability estimates with soft-label BCE and encourage high-success selections with a smooth empirical-regret objective:

$$\mathcal{L}_{\mathrm{BCE}} = \frac{1}{|B|N} \sum_{x \in B} \sum_{i=1}^{N} \mathrm{BCE}\big(q_i(x), \hat{p}_i(x)\big), \tag{8}$$

$$\mathcal{L}_{\mathrm{regret}} = \frac{1}{|B|} \sum_{x \in B} \left[\max_i \hat{p}_i(x) - \sum_{i=1}^{N} \pi_i(x) \hat{p}_i(x) \right], \qquad \pi(x) = \mathrm{softmax}\big(z(x)/\tau_R\big). \tag{9}$$

Here, q_i estimates an expert's success probability, while π_i is a selection probability used only in the training objective. The temperature $\tau_R > 0$ controls the smooth selection distribution; the success probabilities need not sum to one. Continuous regret measures the success-rate gap between this smooth selection and the empirically best expert, encouraging greater selection weight on experts with higher empirical success rates.

To complement these objectives, we directly supervise all pairwise differences. Choosing a weaker expert incurs a loss that grows with its success-rate gap to the better alternative. We therefore supervise all pairwise differences and assign extra weight to comparisons involving the router's current top choice, in proportion to their empirical gaps. Let $t_\theta(x) = \arg\max_i z_i(x)$ and define

$$w_{ij}(x) = 1 + \lambda |\Delta_{ij}(x)| \mathbf{1}\big[t_\theta(x) \in \{i, j\}\big], \tag{10}$$

$$\mathcal{L}_{\mathrm{cmp}} = \frac{\sum_{x \in B} \sum_{i<j} w_{ij}(x) \big[d_{ij}(x) - \Delta_{ij}(x)\big]^2}{\sum_{x \in B} \sum_{i<j} w_{ij}(x)}. \tag{11}$$

The top-choice index and weights are detached from the computation graph. The coefficient $\lambda \geq 0$ controls the additional emphasis on these comparisons. The base weight retains supervision for all pairs; the extra term covers both beneficial alternatives and comparisons that support retaining a better incumbent. Normalization is performed across the entire batch.

The soft selection distribution π is used only during training. **At inference**, the Observer encodes the query into representations. The router uses these representations to predict each expert's success probability. The expert with the highest probability answers the query. Only the selected expert is then invoked to generate an answer.

4 Experiments

We evaluate overall routing performance and query-level routing behavior, assess the three design choices through controlled ablations, and examine token usage and cross-dataset transfer.

Table 1: Absolute success rates (%) under five-expert capability-only routing. All learned routers use OOF evaluation. Best single and Oracle are post-hoc references.

Benchmark	Best single	RouterBench-KNN	RouterBench-MLP	RouteLLM-MF	RouteLLM-SW	Shared TrunkNet	**Full method**	Oracle
SuperGPQA	20.772	21.075	21.475	21.016	20.530	21.702	**23.223**	41.532
LiveCodeBench-v6	39.678	39.033	39.526	39.469	39.678	39.791	**40.645**	45.858
MMLU-Pro	56.848	61.260	61.988	61.677	57.244	62.103	**62.912**	75.986

4.1 Experimental setup

Benchmarks and experts. We evaluate on SuperGPQA (Team et al., 2025), LiveCodeBench-v6 (Jain et al., 2024), and MMLU-Pro (Wang et al., 2024), covering challenging knowledge reasoning, code generation, and multidisciplinary knowledge question answering, with 7,050, 1,055, and 12,032 queries, respectively. The expert pool comprises DeepSeek-R1-Distill-Qwen-7B, Granite-3.3-8B, Qwen2.5-7B-Instruct, Qwen2.5-Coder-7B-Instruct, and Qwen2.5-Math-7B-Instruct. Each expert produces five offline responses per query, evaluated using the corresponding benchmark's correctness criterion. Their correct fraction gives the empirical success label $\hat{p}_i(x)$; their mean input-plus-output token count estimates the usage of one expert invocation. These five samples construct the offline response bank; deployment invokes only the selected expert to generate one answer.

Training and evaluation. In-domain results use benchmark-specific five-fold out-of-fold (OOF) evaluation, with each query excluded from its predictor's training and model selection. The frozen Observer is Qwen3-8B, using final-token residual states from blocks 18–36. Appendix B provides the full training and evaluation protocol.

Metrics and references. For a router R selecting expert $t_R(x)$ on evaluation set $\mathcal{D}$, we report absolute success rate

$$A_R = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \hat{p}_{t_R(x)}(x). \tag{12}$$

The best-single and query-level oracle references are

$$A_{\mathrm{BS}} = \max_i \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \hat{p}_i(x), \qquad A_{\mathrm{oracle}} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \max_i \hat{p}_i(x). \tag{13}$$

Best single is a post-hoc reference on the evaluation set and does not participate in router training or model selection. The oracle is an empirical upper bound on the fixed five-sample response bank. Rates are reported as percentages and differences in percentage points (pp). For token usage and routing behavior, we use the best-single reference, which is DeepSeek-R1-Distill-Qwen-7B (the *thinking* expert) on all three benchmarks.

Baselines. We compare RouterBench-KNN and RouterBench-MLP (Hu et al., 2024), RouteLLM-MF and RouteLLM-SW (Ong et al., 2025), and Shared TrunkNet (Varshney et al., 2026). All methods share the queries, expert pool, five-sample labels, outer folds, and inner data splits. Appendix C documents their implementations, training budgets, and adaptations to five-expert selection.

4.2 Overall routing performance

Routing success. Table 1 evaluates whether query-dependent selection improves over the best-single reference and learned routing baselines. CompareRoute achieves the highest routing success on all three benchmarks. Paired improvements over Shared TrunkNet, the strongest evaluated baseline on each dataset, are 1.521, 0.853, and 0.810 pp, averaging 1.06 pp. Their respective 95% query-bootstrap intervals are [1.101, 1.946], [0.138, 1.568], and [0.562, 1.059] pp. Statistical details appear in Appendix D.1. These results support the effectiveness of CompareRoute in improving routing success across the evaluated benchmarks.

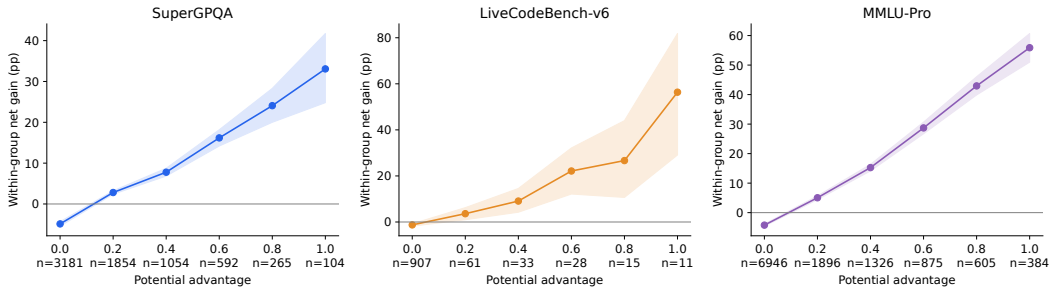

Figure 4: Realized routing gains versus available expert advantage. CompareRoute captures part of the available advantage, achieving larger gains at higher Δ^* while incurring small mean losses at $\Delta^* = 0$. Shading shows pointwise 95% CI.

Table 2: Controlled design comparisons. Values are success-rate differences (pp), first design minus second, with pointwise 95% intervals from 4,000 paired query-bootstrap resamples. Full dynamic denotes expert-conditioned dynamic layer aggregation.

Contrast	SuperGPQA	LiveCodeBench-v6	MMLU-Pro
Shared static − uniform aggregation	$+0.440$ $[0.267, 0.621]$	$+0.701$ $[0.284, 1.176]$	$+0.489$ $[0.386, 0.592]$
Full dynamic − expert-shared dynamic	$+0.377$ $[0.142, 0.624]$	$+1.725$ $[0.891, 2.616]$	$+0.445$ $[0.297, 0.595]$
Full dynamic − expert-specific static	$+0.499$ $[0.255, 0.749]$	$+1.308$ $[0.626, 2.009]$	$+0.424$ $[0.256, 0.592]$
Full − parameter-matched Direct	$+0.624$ $[0.394, 0.879]$	$+1.308$ $[0.569, 2.066]$	$+0.502$ $[0.331, 0.678]$
Weighted − uniform comparison loss	$+0.440$ $[0.235, 0.638]$	$+0.986$ $[0.436, 1.611]$	$+0.475$ $[0.342, 0.612]$

Routing behavior. Figure 4 examines how much of the available expert advantage CompareRoute converts into realized routing gains. We group OOF queries by the empirical success-rate gap between the best expert for each query and the best-single reference, Δ^*, and report CompareRoute's mean gain over that reference within each group. CompareRoute captures substantial routing gains when better experts are available, while incurring mean losses below 5 pp on all three benchmarks when no expert improves on the reference ($\Delta^* = 0$). Appendix E provides the formal definitions and gain–loss decomposition.

4.3 Mechanism Ablations

We test the three design choices in method order: how Observer evidence is aggregated, how experts are compared, and how supervision emphasizes the final choice. All variants are trained from scratch using the same splits, preprocessing, training budget, and model-selection protocol.

Table 2 reports paired success-rate differences and pointwise query-bootstrap intervals conditional on fixed OOF selections and the response bank. Appendix D.2 gives the statistical protocol.

Expert-conditioned dynamic aggregation. Three controls distinguish learning layer weights, conditioning on expert identity, and adapting to the query. Shared static aggregation outperforms uniform aggregation on all datasets, showing the value of learning which layers to emphasize. Full dynamic aggregation further improves over expert-shared dynamic aggregation, supporting expert conditioning while retaining query adaptation. It also improves over expert-specific static aggregation, supporting query adaptation while retaining expert-specific weights. Together, these comparisons support reading the shared Observer evidence according to both the query and the expert.

Explicit expert comparison. We compare Full with a Direct scorer that uses the same Observer features, dynamic aggregation, and supervision labels, including expert-difference labels. Direct predicts each expert's success probability directly, whereas Full first predicts pairwise success-rate differences and aggregates them into expert scores. Full improves routing success by 0.624, 1.308, and 0.502 pp on SuperGPQA, LiveCodeBench-v6, and MMLU-Pro, respectively. These gains support the effectiveness of our comparison design for routing. Appendix F.1 details the two scoring designs.

Decision-relevant learning. We compare our weighting scheme with equal weighting of all expert pairs. The architecture and other losses remain unchanged. Our weighting improves success by 0.440, 0.986, and 0.475 pp. These gains support giving more weight to pairs involving the current top choice. The extra weight increases with the empirical success-rate gap.

Core module ablations across repeats. Table 3 complements these controlled comparisons with five independent experimental repeats of whole-module removals. Within each repeat, we pair every variant with Full under the same splits and evaluation protocol, then report the mean and standard deviation of the paired differences. Removing any individual module lowers mean success on all three benchmarks; removing all three yields the largest drops, of 0.610, 1.611, and 0.889 pp, respectively. Appendix F.2 specifies the replacements and retained losses.

Table 3: Module ablations: success-rate changes relative to Full (pp), reported as mean ± standard deviation of paired differences over five independent experimental repeats. Negative values indicate degradation.

Model variant	SuperGPQA	LiveCodeBench-v6	MMLU-Pro
Without dynamic layer aggregation	-0.318 ± 0.103	-0.701 ± 0.365	-0.322 ± 0.058
Without expert comparison	-0.170 ± 0.131	-1.365 ± 0.402	-0.492 ± 0.105
Without decision optimization	-0.216 ± 0.115	-0.872 ± 0.315	-0.607 ± 0.112
Without all three modules	-0.610 ± 0.204	-1.611 ± 0.407	-0.889 ± 0.113

4.4 Token usage and cross-dataset transfer

Selected-expert token usage. We measure token usage under the capability policy. The best-single reference always invokes DeepSeek-R1-Distill-Qwen-7B. CompareRoute achieves success-rate gains of 2.45, 0.97, and 6.06 pp and token reductions of 34.77%, 20.49%, and 36.70%, averaging 30.65% across benchmarks. These are resource-use outcomes of capability-based selection. We also explored incorporating token usage into the training loss, but observed no benefits.

Observer overhead and net token savings. Routing requires one prefill pass through the frozen Observer, with mean input lengths of 394.0, 534.5, and 241.8 tokens on SuperGPQA, LiveCodeBench-v6, and MMLU-Pro, respectively, and no Observer decoding. Including these inputs, total token usage is 4,957.1, 5,967.7, and 2,312.5 tokens per query. Compared with always invoking the best-single expert on each benchmark, these totals represent net reductions of 29.14%, 12.67%, and 29.31%, averaging 23.71% across benchmarks (Figure 5). These savings accompany success-rate improvements of 2.45, 0.97, and 6.06 percentage points, respectively. Our accounting includes every Observer input token without discounting prefill, although the additional routing pass requires no decode generation.

Cross-dataset transfer. Figure 6 evaluates cross-dataset transfer with the source preprocessing and routing networks frozen and no target adaptation or model selection. Within each target, transfers, in-domain OOF results, and expert references use the same deduplicated query set. Transfer retains gains over the target best single from MMLU-Pro to SuperGPQA (+1.19 pp) and from SuperGPQA to MMLU-Pro (+2.70 pp), but none of the four directions involving LiveCodeBench-v6 exceeds this reference. All six transfer point estimates fall below their corresponding in-domain OOF results. These results support partial retention of routing gains between the two knowledge QA benchmarks, while exposing a limitation in transfer across the evaluated code and knowledge QA tasks. Appendix G gives the complete success rates, evaluation protocol, and statistical analysis.

5 Discussion and Limitations

Adaptive use of Observer information. SharedTrunkNet (Varshney et al., 2026) has demonstrated the value of LLM prefill hidden states for predicting candidate-model correctness. Building on this finding, our results suggest that the Observer's multilayer hidden states can be used more effectively through dynamic aggregation conditioned on both the query and expert identity. Our controlled aggregation comparisons support both forms of conditioning, with gains over expert-shared dynamic

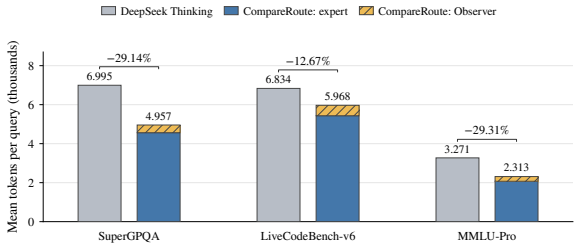

Figure 5: Mean token usage per query versus Best-Single, including Observer input tokens. Labels show net token reductions

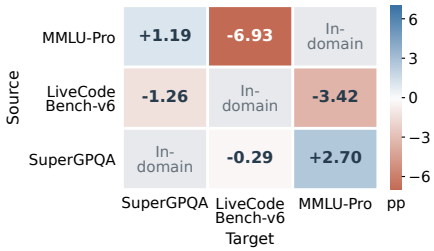

Figure 6: Frozen transfer gain over target best single (pp), on matched, deduplicated target sets.

and expert-specific static aggregation. These findings highlight the importance of learning which Observer information is relevant to each query–expert pair.

Decision-relevant relative capability. Expert complementarity creates opportunities for routing, but exploiting them requires reliable judgments of relative advantage. With finite data and model capacity, reducing per-expert success-probability prediction error alone does not guarantee better expert selection. Learning should therefore prioritize comparisons that affect the final expert choice, especially when selecting a weaker expert would incur a large success-rate loss. Concretely, CompareRoute predicts pairwise success-rate differences with a shared antisymmetric comparator and aggregates these predictions into relative scores for selection. Its comparison loss assigns additional weight to pairs involving the router's current top choice, proportional to their empirical success-rate gaps. The controlled comparisons in Section 4.3 support this direction, suggesting that organizing capability estimates and their supervision around expert selection can help translate complementarity into realized routing gains.

Evaluation scope. The current evidence is based on a fixed five-expert pool and a single Observer; The transfer results in Section 4.4 show that routing gains do not consistently carry over across the evaluated tasks. Generalization across datasets and expert pools remains an open challenge. Minimax-regret training offers a potential direction for improving robustness to unknown mixtures of known source domains (Dann et al., 2025).

6 Conclusion

We presented CompareRoute, a pre-generation router that explicitly models relative expert capability using the multilayer representations of a shared frozen Observer. The method combines expert-conditioned dynamic layer aggregation, antisymmetric expert comparison, and decision-focused training; inference requires one Observer prefill and invokes only the selected expert for answer generation. Across SuperGPQA, LiveCodeBench-v6, and MMLU-Pro with five heterogeneous experts, in-domain out-of-fold evaluation shows an average success-rate improvement of 1.06 percentage points over Shared TrunkNet, the strongest evaluated baseline on each benchmark. Including Observer input tokens, total token usage decreases by an average of 23.71% across benchmarks relative to the best-single expert reference. These results support learning relative expert capability from shared pre-generation evidence to improve routing.

AI USE STATEMENT

We used generative AI tools to assist with method implementation, result interpretation, and translation. Additionally, we used these tools for code drafting and manuscript revision. The authors reviewed the AI-assisted outputs and take full responsibility for the final manuscript, code, and reported findings.

ETHICS STATEMENT

This work evaluates LLM routing on existing benchmarks. Routed outputs may inherit biases and harmful behaviors from the underlying models; optimizing benchmark correctness does not ensure safe or fair outputs.

REPRODUCIBILITY STATEMENT

Sections 3 and 4.1 describe the method and evaluation setup. Appendices A–D provide architectural details, preprocessing, training and model-selection settings, baseline adaptations, and statistical procedures.

REFERENCES

Iván Vicente Moreno Cencerrado, Arnau Padrés Masdemont, Anton Gonzalvez Hawthorne, David Demitri Africa, and Lorenzo Pacchiardi. No answer needed: Predicting llm answer accuracy from question-only linear probes, 2026. URL https://arxiv.org/abs/2509.10625.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James T. Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models, 2024. URL https://arxiv.org/abs/2409.19886.

Christoph Dann, Yishay Mansour, Teodor Vanislavov Marinov, and Mehryar Mohri. Principled model routing for unknown mixtures of source domains. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=w7Fe5rHLvJ.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing, 2024. URL https://arxiv.org/abs/2404.14618.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections, 2025. URL https://arxiv.org/abs/2410.03834.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. Routerbench: A benchmark for multi-llm routing system, 2024. URL https://arxiv.org/abs/2403.12031.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Guannan Lai and Han-Jia Ye. When routing collapses: On the degenerate convergence of llm routers, 2026. URL https://arxiv.org/abs/2602.03478.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, Zhen Wang, and Shuyue Hu. Llmrouterbench: A massive benchmark and unified framework for llm routing, 2026. URL https://arxiv.org/abs/2601.07206.

Yang Li. Rethinking predictive modeling for llm routing: When simple knn beats complex learned routers, 2026. URL https://arxiv.org/abs/2505.12601.

William Lugoloobi, Thomas Foster, William Bankes, and Chris Russell. Llms encode their failures: Predicting success from pre-generation activations, 2026. URL https://arxiv.org/abs/2602.09924.

Jayanta Mandi, Víctor Bucarey, Maxime Mulamba, and Tias Guns. Decision-focused learning: Through the lens of learning to rank, 2022. URL https://arxiv.org/abs/2112.03609.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2025. URL https://arxiv.org/abs/2406.18665.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. IRT-router: Effective and interpretable multi-LLM routing via item response theory. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15629–15644, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.761. URL https://aclanthology.org/2025.acl-long.761/.

M-A-P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, Kaixin Deng, Shawn Gavin, Shian Jia, Sichao Jiang, Yiyan Liao, Rui Li, Qinrui Li, Sirun Li, Yizhi Li, Yunwen Li, David Ma, Yuansheng Ni, Haoran Que, Qiyao Wang, Zhoufutu Wen, Siwei Wu, Tyshawn Hsing, Ming Xu, Zhenzhu Yang, Zekun Moore Wang, Junting Zhou, Yuelin Bai, Xingyuan Bu, Chenglin Cai, Liang Chen, Yifan Chen, Chengtuo Cheng, Tianhao Cheng, Keyi Ding, Siming Huang, Yun Huang, Yaoru Li, Yizhe Li, Zhaoqun Li, Tianhao Liang, Chengdong Lin, Hongquan Lin, Yinghao Ma, Tianyang Pang, Zhongyuan Peng, Zifan Peng, Qige Qi, Shi Qiu, Xingwei Qu, Shanghaoran Quan, Yizhou Tan, Zili Wang, Chenqing Wang, Hao Wang, Yiya Wang, Yubo Wang, Jiajun Xu, Kexin Yang, Ruibin Yuan, Yuanhao Yue, Tianyang Zhan, Chun Zhang, Jinyang Zhang, Xiyue Zhang, Xingjian Zhang, Yue Zhang, Yongchi Zhao, Xiangyu Zheng, Chenghua Zhong, Yang Gao, Zhoujun Li, Dayiheng Liu, Qian Liu, Tianyu Liu, Shiwen Ni, Junran Peng, Yujia Qin, Wenbo Su, Guoyin Wang, Shi Wang, Jian Yang, Min Yang, Meng Cao, Xiang Yue, Zhaoxiang Zhang, Wangchunshu Zhou, Jiaheng Liu, Qunshu Lin, Wenhao Huang, and Ge Zhang. Supergpqa: Scaling llm evaluation across 285 graduate disciplines, 2025. URL https://arxiv.org/abs/2502.14739.

Asterios Tsiourvas, Wei Sun, and Georgia Perakis. Causal llm routing: End-to-end regret minimization from observational data, 2025. URL https://arxiv.org/abs/2505.16037.

Tanay Varshney, Annie Surla, Michelle Xu, Gomathy Venkata Krishnan, Maximilian Jeblick, David Austin, Neal Vaidya, and Davide Onofrio. Llm router: Rethinking routing with prefill activations, 2026. URL https://arxiv.org/abs/2603.20895.

Xuanhui Wang, Cheng Li, Nadav Golbandi, Mike Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In *Proceedings of The 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*, pp. 1313–1322, 2018.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

A Additional router details

A.1 Pair-conditioned context and comparison inputs

For each unordered expert pair (i, j), we construct a common comparison context from the expert-conditioned summaries $s_i(x), s_j(x) \in \mathbb{R}^d$ defined in Eq. (3). We omit the query argument below for readability. A separate embedding table $e_i^{\text{pair}} \in \mathbb{R}^{d_e}$ encodes expert identity for comparison and is learned independently of the aggregation embeddings e_i^{expert}. We form a symmetric pair condition

$$q_{ij} = W_q\big[e_i^{\text{pair}} + e_j^{\text{pair}}; |e_i^{\text{pair}} - e_j^{\text{pair}}|\big], \tag{14}$$

where brackets denote concatenation and the absolute value is elementwise. Attention over the two summaries produces

$$\eta_{ij,k} = w_c^\top \tanh(W_c s_k + q_{ij}), \qquad k \in \{i, j\}, \tag{15}$$

$$\beta_{ij,k} = \frac{\exp(\eta_{ij,k})}{\sum_{v\in\{i,j\}} \exp(\eta_{ij,v})}, \qquad c_{ij} = \beta_{ij,i} s_i + \beta_{ij,j} s_j. \tag{16}$$

Swapping the expert order preserves q_{ij} and exchanges the summaries and their weights, so $c_{ij} = c_{ji}$.

The ordered input to the shared comparator concatenates the context-centered summaries, the context itself, and the expert embeddings:

$$\xi_{ij} = [s_i - c_{ij};\ s_j - c_{ij};\ c_{ij};\ e_i^{\text{pair}};\ e_j^{\text{pair}}] \in \mathbb{R}^{3d+2d_e}. \tag{17}$$

The input ξ_{ji} uses the reverse expert order and the same context. Both inputs are evaluated by the same two-layer MLP f_θ, with a GELU hidden activation, to obtain the antisymmetric difference in Eq. (4).

A.2 Shared offset head

The concatenated summaries $S(x) = [s_1(x); \ldots; s_N(x)] \in \mathbb{R}^{Nd}$ are passed through a shared MLP T_θ with GELU activations and dropout:

$$[o_1, \ldots, o_N] = T_\theta(S(x)). \tag{18}$$

The N outputs are averaged into the common logit offset used in Eq. (7):

$$b(x) = \frac{1}{N} \sum_{i=1}^{N} o_i(x). \tag{19}$$

The offset is learned through success supervision without a separate difficulty label.

A.3 Network widths, dropout, and initialization

The same architecture is used on all three benchmarks, with $N = 5$ experts, $L = 19$ Observer layers, summary width $d = 100$, and identity/attention width $d_e = 32$. Table 4 specifies every trainable network width. All linear maps in the layer-attention and pair-context attention modules are bias-free. The per-layer projections and shared-trunk linear layers have biases; the comparator has a hidden-layer bias and no output bias.

Table 4: Complete router architecture. Widths exclude the frozen Observer and PCA. The two identity tables for expert aggregation and comparison are independent.

Component	Dimensions and operations
Layer projections	19 separate $32 \to 100$ linear maps, each followed by GELU
Aggregation embeddings	Layer table 19×32; expert table 5×32
Layer-attention scorer	$W_g : 100 \to 32$; additive embeddings, $\tanh$, then $w_g : 32 \to 1$
Comparison embeddings	Separate expert table 5×32
Pair-context attention	$W_c : 100 \to 32$; $W_q : 64 \to 32$; $\tanh$, then $w_c : 32 \to 1$
Shared comparator f_θ	$364 \to 64 \to 1$, with GELU after the 64-unit layer; $364 = 3d + 2d_e$
Shared trunk	$500 \to 256 \to 128$, with GELU and dropout after each hidden layer: rates 0.2 and 0.1, respectively
Output heads	Two $128 \to 5$ linear heads: offset logits o_i; implemented together as $128 \to 10$
Probability scale	One learned scalar γ, with $a = \text{softplus}(\gamma) + 10^{-6}$

The first five outputs of the shared trunk define T_θ in Eq. (18); their mean is $b(x)$. The other five outputs are excluded from capability-only selection. Dropout is used only at the two shared hidden layers and is disabled at evaluation; the layer projections, attention modules, and comparator have no dropout. We initialize the comparator's final weight matrix to zero and $\gamma = \log(\exp(4) - 1)$, giving an initial scale $a = 4 + 10^{-6}$. Pair-context attention uses an unscaled softmax (temperature 1).

B ADDITIONAL TRAINING DETAILS

Data splits and evaluation. For in-domain evaluation, we train a separate router for each benchmark using five outer folds, with each outer training set divided into fit and checkpoint subsets at an 85:15 ratio. Preprocessing is fitted on outer-training features; only the fit subset updates the network, and only the checkpoint subset selects models. Every evaluation query is therefore excluded from both network training and model selection for its predictor. We concatenate the five held-out folds to obtain out-of-fold (OOF) selections, then aggregate results over queries.

Observer features and preprocessing. The frozen Qwen3-8B Observer supplies final-token residual states from all 19 blocks in the range 18–36. Per-layer PCA retains 32 dimensions, projected layer vectors have width 100, and identity embeddings have width 32. PCA and standardization are fitted on the corresponding outer training set and remain fixed during network training and evaluation.

Optimization and model selection. For each outer fold, we train ten initialization seeds with Adam, a learning rate of 10^{-3}, batch size 64, at most 500 epochs, and early-stopping patience 30. The checkpoint criterion maximizes the sum of empirical success rates of the capability-only top choices on the inner validation set, breaking ties by lower capability BCE. We retain five initializations by that criterion and average their success-probability predictions for inference. The retained routers share the same Observer prefill and preprocessing.

Temperatures and comparison weighting. We use $\tau_L = 1.0$ in Eq. (3), $\tau_R = 0.5$ in Eq. (9), and $\lambda = 2.0$ in Eq. (10) on all three benchmarks. Thus, a pair involving the current top choice receives weight $1 + 2|\Delta_{ij}|$, ranging from 1 to 3; every other pair has weight 1. For example, an empirical gap of 0.2 gives a weight of 1.4 for a pair involving the top choice. The uniform-comparison control uses $w_{ij} = 1$ for every pair, equivalently $\lambda = 0$, while retaining the same temperatures and loss coefficients.

Routing objective. The capability and decision terms form the routing objective

$$\mathcal{L}_{\text{route}} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{regret}} + \mathcal{L}_{\text{cmp}}, \tag{20}$$

where $\mathcal{L}_{\text{BCE}}$, $\mathcal{L}_{\text{regret}}$, and $\mathcal{L}_{\text{cmp}}$ are defined in Eqs. (8), (9), and (11), respectively, each with coefficient 1. All trainable router components are optimized jointly while the Observer and preprocessing remain fixed. The losses $\mathcal{L}_{\text{regret}}$ and $\mathcal{L}_{\text{cmp}}$ together constitute the decision optimization objective in the module ablations.

B.1 EXPERT GENERATION AND CORRECTNESS EVALUATION

Response bank and decoding. The five frozen expert checkpoints are listed in Section 4.1. SuperGPQA uses the complete official hard partition (7,050 queries); LiveCodeBench-v6 and MMLU-Pro contain 1,055 and 12,032 queries, respectively. For each available query–expert pair, we collect $K = 5$ responses (sample IDs 0–4) using vLLM 0.10.2 in bfloat16, with sampling temperature 0.6, top-p 0.95, and a maximum of 8,192 generated tokens. The served context limit is 16,384 input-plus-output tokens for every expert except Qwen2.5-Math-7B-Instruct, which uses its native 4,096-token limit; the remaining context therefore also bounds output length. The five completions are drawn within one generation request, seeded with $20{,}270{,}809 + j$ for the query's zero-based standardized index j. Other sampling settings use the vLLM defaults, with no custom stop strings.

Prompt construction. Each prompt is supplied as one user message through the expert's own tokenizer chat template, with `add_generation_prompt=True`. We supply no separate system message or few-shot examples; any default system text comes from the native template. For both multiple-choice benchmarks, the instruction is:

> Answer the following multiple-choice question. Explain your reasoning, then finish with `Answer: X`, where X is one of *[comma-separated option letters]*.

We append the question and its options in their original order, with one option per line, labeled `A.`, `B.`, and so on. For LiveCodeBench-v6, the instruction is:

> You are an expert Python programmer. Solve the following programming problem so that the program passes all tests. You may reason before the final code.

After the problem text, tasks with starter code receive "Use the following starter code and return the completed solution inside a Python code block," followed by that starter code in a Python fence. Tasks without starter code instead receive "Read from stdin and write to stdout. Return the complete solution inside a Python code block."

Multiple-choice grading. For SuperGPQA and MMLU-Pro, a local deterministic parser extracts an option letter and compares it exactly with the gold letter. The parser accepts explicit `answer`/`choice` markers, letters inside `\boxed` or `\fbox`, and a trailing standalone letter. It selects the candidate appearing latest in the response and normalizes the letter to uppercase. An absent or incorrect answer receives a binary score of zero. The response bank uses parser version `explicit-or-boxed-choice-v2-2026-08-21`.

Code grading. For LiveCodeBench-v6, we extract the last fenced code block and run the official `codegen_metrics` evaluator on the combined public and private tests. Python execution takes place in an isolated Docker container with the evaluator timeout set to 6 seconds. A response receives a score of one only if every test passes; missing code, compilation or runtime errors, failed tests, and timeouts receive zero. No learned judge is used for any benchmark.

Labels, token counts, and availability. We average the five binary correctness scores to obtain $\hat{p}_i(x)$; this estimates single-response success under the specified sampling protocol. Token usage is the mean number of input-plus-output tokens across those same five responses, measured with each expert's tokenizer and including reasoning tokens. Length-limited responses are evaluated as returned. One SuperGPQA prompt (standardized index 2374) exceeds Qwen2.5-Math's context limit: its five entries are recorded as unavailable, with empty responses and zero correctness, and that expert is masked for this query. All other query–expert pairs in the three benchmarks are available. Our router masks unavailable entries in BCE, masks unavailable pairs in comparison loss, and restricts regret, relative-score recovery, shared-offset averaging, and routing to available experts, normalizing each expression over the corresponding available entries or pairs. When all experts are available, these expressions reduce to the equations in the main text. All reported methods are evaluated against the same fixed response bank; evaluating a router reads the selected expert's five-sample success fraction, while deployment generates one response from the selected expert.

C Baseline implementation and adaptation details

We implement the RouterBench baselines (Hu et al., 2024), RouteLLM variants (Ong et al., 2025), and Shared TrunkNet (Varshney et al., 2026) as described below, adapting them to the common five-expert protocol.

RouterBench-KNN. We use frozen MiniLM query embeddings and cosine KNN regression to predict the five experts' empirical success rates. The checkpoint subset selects the number of neighbors from 5, 10, and 40.

RouterBench-MLP. The same embeddings feed two ReLU hidden layers of 100 units each to predict five expert success rates, trained with MSE.

RouteLLM-MF. We adapt matrix-factorization scoring to five-expert selection. A query projection interacts with normalized expert vectors, trained using expert-pair Bradley–Terry BCE with rank 128.

RouteLLM-SW. Query similarity weights expert preferences in the fit set. We solve for a Bradley–Terry ranking and select the highest-scoring expert, with similarity-weight parameter $\gamma = 10$.

Shared TrunkNet. We follow Shared TrunkNet's original training and ensemble-selection procedure, using static Observer activations with per-expert layers selected by Fisher scores. A shared

multi-output predictor is trained with BCE, and predictions from five validation-selected seeds are averaged for routing.

All baselines share the full method's evaluation protocol, query sets, five-expert pool, five-sample labels, five outer folds, and inner fit/checkpoint splits. MLP and MF likewise train 10 seeds per fold, retain five, and average predictions; KNN and SW build their retrieval or preference banks using only fit data. RouteLLM's two-model preference routing is extended to five-expert scoring. Training preferences are constructed from empirical success-rate comparisons, with ties assigned 0.5, and a local MiniLM replaces the original encoder.

D Statistical analyses

D.1 In-domain comparisons with Shared TrunkNet

Section 4.2 reports paired success-rate differences and confidence intervals against Shared TrunkNet, the strongest evaluated baseline on each benchmark. We specify their construction here.

We use a query-level paired bootstrap, holding both methods' OOF selections fixed and aligning queries by ID under the current five-sample labels. We draw 10,000 bootstrap resamples with replacement (seed 20260909) and report percentile 95% confidence intervals for paired success-rate differences.

D.2 Module repeats and controlled design comparisons

Table 3 reports five independent experimental repeats. Within each repeat, one experimental seed runs Full and every ablation with the same data splits and evaluation protocol. We compute each variant's success-rate change relative to Full within that repeat, then report the mean and standard deviation of the five paired differences.

For each contrast in Table 2, we instead hold both models' OOF selections and the five-sample response bank fixed and compute per-query differences between the selected experts' empirical success rates. We draw 4,000 paired bootstrap resamples, each containing the original number of queries sampled with replacement. The same query indices are used for both sides of a contrast. We recompute the mean difference for each resample and take its 2.5th and 97.5th percentiles as the interval endpoints.

These pointwise intervals quantify query-resampling variation conditional on the fixed predictions and response bank. They exclude variation from training, regenerating responses, and constructing selections, and are computed separately without multiple-comparison correction.

E Routing behavior analysis

We define the quantities used in Figure 4. The analysis uses fixed OOF selections and the same five-sample response bank, with DeepSeek-R1-Distill-Qwen-7B as the thinking reference on all three benchmarks.

Realized routing gain. For each query x, the router's realized gain relative to the thinking reference is

$$\delta_R(x) = \hat{p}_{t_R(x)}(x) - \hat{p}_{\text{thinking}}(x).$$

A positive value means that the selected expert has a higher empirical success rate than the thinking reference; a negative value indicates a harmful choice.

Available improvement groups. The available empirical improvement over the thinking reference is

$$\Delta^*(x) = \max_i \hat{p}_i(x) - \hat{p}_{\text{thinking}}(x).$$

The five-sample labels yield six possible values of $\Delta^*(x)$, from 0 to 1 in increments of 0.2. Figure 4 averages $\delta_R(x)$ within each group, using that group's queries as the denominator. A group's contribution to the full-dataset gain therefore also depends on its frequency.

MMLU-Pro captures larger gains in the high-advantage groups than SuperGPQA; the corresponding LiveCodeBench-v6 estimates are less precise, with only 15 and 11 queries in its two highest-advantage groups. These patterns describe where CompareRoute succeeds and leaves opportunities uncaptured.

F Additional ablation results and implementation details

F.1 Parameter-matched Direct scorer

To evaluate the contribution of explicit expert comparison while controlling for differences in parameter count, we construct a nearly parameter-matched direct scorer (Direct). Both models use the same Observer features, dynamic layer aggregation architecture, supervision labels, data splits, training budgets, and model-selection protocol. Full first predicts pairwise differences in expert success rates and then aggregates them into expert scores, whereas Direct predicts each expert's success probability directly. This control compares the routing performance of the two scoring designs under approximately matched parameter budgets.

As shown in Table 2, Full improves routing success over Direct by 0.624, 1.308, and 0.502 percentage points on SuperGPQA, LiveCodeBench-v6, and MMLU-Pro, respectively. These results show that explicit comparison outperforms this direct scorer on all three benchmarks under the matched conditions, supporting its effectiveness for expert routing.

F.2 Core module ablations

Table 3 in Section 4.3 reports the module-removal results. Here we specify the replacements and retained losses. These removals assess each module within the full design: deleting expert comparison removes its structure and comparison supervision, while deleting decision optimization removes both continuous regret and comparison supervision.

Without dynamic layer aggregation. We remove the input-dependent, expert-conditioned dynamic layer-gating parameters. We retain all 19 Observer layers (18–36) and per-layer projections, fixing each aggregation weight to $1/19$. Expert comparison, relative-score recovery, and the success-probability mapping remain unchanged. The retained objective is capability BCE plus continuous regret and decision-relevant comparison loss.

Without expert comparison. We retain dynamic layer aggregation and remove expert-pair representations, comparison attention, the comparison network, comparison supervision, and relative-score recovery. A shared prediction network directly outputs each expert's success probability; routing uses the same probability-averaging rule. The retained objective is capability BCE plus continuous regret.

Without the decision optimization objective. We retain the full routing architecture, including dynamic layer aggregation, expert comparison, relative-score recovery, and the success-probability mapping, while removing both continuous regret and the decision-relevant comparison auxiliary loss. The comparison branch still receives gradients through the final capability BCE. The retained objective is capability BCE.

Without all three modules. We use uniform aggregation over blocks 18–36, remove the entire comparison branch, predict expert success probabilities directly, and remove both decision optimization losses. The retained objective is capability BCE.

G Cross-dataset transfer

This appendix provides the complete transfer results and details of target-set construction and statistical analysis. Source preprocessing and routing networks remain frozen, with no target-data training or model selection.

Matched target sets and complete results. A common lexical deduplication rule excludes 324 SuperGPQA and 344 MMLU-Pro queries, leaving target sets of 6,726, 1,055, and 11,688 queries

for SuperGPQA, LiveCodeBench-v6, and MMLU-Pro, respectively. Transfer, in-domain OOF, and expert-reference results use identical target queries; source models are not retrained. Table 5 reports the complete success rates.

Table 5: Frozen-router success rates (%) on common deduplicated target sets (rows: training sources; columns: evaluation targets). Bold diagonal entries are recomputed in-domain OOF results and may differ from Table 1. Target best single is a post-hoc reference.

Training source	SuperGPQA	LiveCodeBench-v6	MMLU-Pro
SuperGPQA	**22.64**	39.39	60.48
LiveCodeBench-v6	18.92	**40.64**	54.36
MMLU-Pro	21.37	32.75	**63.60**
Target best single	20.18	39.68	57.78

Statistical analysis. Table 6 reports transfer minus in-domain OOF success rate on the matched target sets. Its pointwise 95% confidence intervals use 2,000 paired cluster-bootstrap resamples, shared by both results. Queries with identical question text or a shared passage are resampled together. The intervals condition on fixed models and the existing response bank.

Table 6: Transfer minus in-domain OOF success rate on matched, deduplicated target queries. Differences and pointwise confidence intervals are in pp.

Transfer direction	Difference	95% CI
MMLU-Pro → SuperGPQA	-1.27	$[-1.65, -0.91]$
LiveCodeBench-v6 → SuperGPQA	-3.72	$[-4.17, -3.29]$
SuperGPQA → LiveCodeBench-v6	-1.25	$[-2.05, -0.44]$
MMLU-Pro → LiveCodeBench-v6	-7.90	$[-9.07, -6.79]$
SuperGPQA → MMLU-Pro	-3.12	$[-3.48, -2.77]$
LiveCodeBench-v6 → MMLU-Pro	-9.24	$[-9.75, -8.75]$