GENERALIZABLE ROUTING TO UNSEEN LLMS VIA DYNAMIC CASE-BASED MEMORY AND DIVERSE CANDIDATE SETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Routing among heterogeneous LLMs is essential for reducing the execution cost of agentic systems, where a router must select the most cost-effective model at each round. Generalizing to unseen LLMs remains difficult: state-of-the-art routers depend on representations or memories that are not adapted to the evolving candidate pool, and they are typically trained on a single large pool that limits exposure to diverse LLM combinations. We introduce GREAT, a case-based memory that stores the most recent routing decision for similar inputs and is dynamically updated so that each decision reflects the latest capabilities of the underlying models. We further train routers on a diverse set of candidate sets simultaneously, deriving a candidate-set selection score from the meta-RL formulation, proving its weak submodularity, and using a greedy algorithm with adaptive stopping to construct near-optimal sets. Experiments across diverse benchmarks demonstrate the effectiveness of our approach.

1 INTRODUCTION

LLM routing, which selects an appropriate LLM out of a set of candidates to invoke for a given input, aims to reduce the execution cost of LLM-based systems without sacrificing output quality (Varangot-Reille et al., 2026a; Moslem & Kelleher, 2026). Because candidate LLMs differ substantially in capability and cost, the router's purpose is to pick the most cost-effective model for each input rather than always defaulting to the strongest or cheapest (Ong et al., 2025; Chen et al., 2024a). The central challenge here is *generalizability*: routers must transfer to unseen LLMs, as new models and APIs continually replace old ones (Wang et al., 2026; Feng et al., 2025). Static routers trained on a fixed pool fail in this setting, degrading sharply when the pool changes (Varangot-Reille et al., 2026b; Moslem & Kelleher, 2026). The problem is even harder in agentic systems, where an agent coordinates multiple models across multi-round planning, tool use, and memory updates, so poor decisions compound over rounds, and the router must sustain effective routing throughout an entire workflow, not just at its first step (Feng et al., 2026; Zhang et al., 2025; Qian et al., 2025).

Existing approaches to generalizable routing fall into two categories. **Memory-based routers**, where memory denotes any store the router reads from via attention, differ primarily in what they store: model capability representations (Jitkrittum et al., 2026; Wang et al., 2026), per-input performance records (Varangot-Reille et al., 2026b), model-level entries aggregated from past executions (Feng et al., 2025; 2026), or static model descriptors such as pricing and latency (Zhang et al., 2025). **Optimization-based routers** instead use bi-level optimization to learn a shared initialization that generalizes to held-out pools (Zeng et al., 2026).

Despite this progress, existing memory-based routers share a structural weakness: **they store information about models rather than about routing decisions, so the retrieved memory does not discriminate among candidates for the current decision.** For instance, GraphPlanner Feng et al. (2026) aggregates historical query-response interactions through shared LLM-role representations, which may entangle evidence from heterogeneous past states. As illustrated in the left of Figure 1, this causes outdated poor models (e.g., C) to receive high weights for similar tasks, leading to suboptimal routing decisions. **Moreover, the optimal decision for the same input query depends on which candidate LLMs are available**, which suggests that memory entries should be stored for different candidate LLM combinations. However, existing methods ignore such dependence:

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.

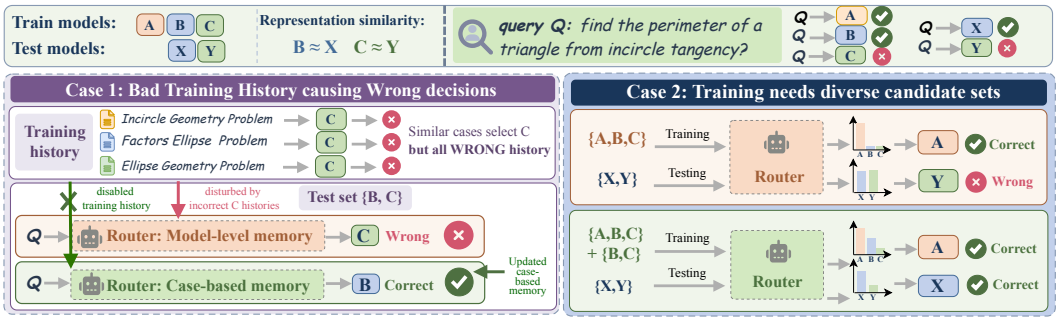


Figure 1: Motivating example of case-based retrieval and diverse candidate set training for generalizable LLM routing (GraphPlanner Feng et al. (2026)). **Left:** GraphPlanner stores all past executions regardless of utility, so poor models that have been used for similar tasks (e.g., C) receive high score. *Our solution: store individual cases and periodically update memory to remove outdated decisions*. **Right:** With a single candidate set, the router assigns high probability only to the best LLM, making low-scoring LLMs hard to distinguish in other sets. *Our solution: train with multiple diverse candidate sets*.

as shown in the right of Figure 1, when trained on a single candidate set, the router assigns high probability only to the best LLM, making low-scoring LLMs hard to distinguish. Hence, this leads to poor generalizability to new candidate sets, e.g., those comprising only low-scoring LLMs.

We propose a framework that addresses both issues. First, we introduce a *case-based memory module that stores each routing decision together with the input context in which it was made*. The stored memory entries are retrieved by input similarity and conditioned on the candidate set during training and inference, so that decisions made under different pools are not conflated. The memory is dynamically updated, pruning outdated entries and refreshing those for similar contexts, so stored decisions track the improving policy and provide increasingly reliable guidance.

Second, since optimal decisions depend on the candidate set, the router must be trained to handle arbitrary candidate sets rather than a single pool. Training on multiple diverse candidate sets is one way to achieve this, but the choice of these sets used for training is critical. *To derive a principled criterion for set selection, we formalize the problem as a bi-level optimization*: the inner loop trains a router on a collection of candidate sets, and the outer loop measures generalization to held-out sets. From this formulation, *we derive a score that measures the alignment between the training gradient on a new candidate set and the meta-gradient on held-out LLM sets*. We prove the score weakly submodular in the candidate set, thus enabling an efficient greedy algorithm to construct near-optimal sets.

Experiments across 11 benchmarks show that GREAT consistently improves existing routers. Under disjoint training and test LLM pools, GREAT improves average accuracy by 1.6–7.1 percentage points while reducing average execution cost by 8.8%–36.5% across five routing backbones. On held-out datasets with unseen LLMs, the corresponding accuracy gains are 3.3–5.2 percentage points, with cost reductions of 8.2%–57.6%.

Our contribution includes:

- **Case-based memory**: stores routing decisions with input context, retrieved by input similarity conditioned on each candidate set, dynamically updated to track routing policy improvement.
- **Diverse candidate LLM set construction**: a weakly submodular score derived from a bi-level formulation, optimized by greedy adaptive stopping.
- **Empirical validation**: consistent accuracy gains and cost reductions across five backbones and 11 benchmarks, including unseen LLMs, unseen candidate-set combinations, and held-out datasets.

2 Related Work

Cost-aware and transferable LLM routing. LLM cascades and learned routers reduce inference cost by selecting models via confidence, difficulty, preferences, or online feedback (Chen et al., 2024a; Aggarwal et al., 2024; Ding et al., 2024; Ong et al., 2025; Shen et al., 2025; Chen et al.,

2024b; Ding et al., 2025; Wang et al., 2025; Jin et al., 2025; Shirkavand et al., 2025; Tsiourvas et al., 2025; Panda et al., 2025; Shah & Shridhar, 2025). Behavioral and profile-based representations enable generalization to unseen models by encoding each model through its responses or benchmark profile rather than an ID (Jitkrittum et al., 2026; Wang et al., 2026; Zhuang et al., 2025; Song et al., 2025; Yan et al., 2026; Xu et al., 2026); benchmark-derived and online-constraint routing provide complementary evidence (Hu et al., 2024; Huang et al., 2025; Guha et al., 2024; Stripelis et al., 2024; Dann et al., 2025; Wu & Silwal, 2025).

Training-set construction for generalizable routing. Existing routers are trained on a fixed pool of models, and generalization is pursued by changing the router's representation (Jitkrittum et al., 2026; Wang et al., 2026; Zhuang et al., 2025; Song et al., 2025; Yan et al., 2026; Xu et al., 2026). *In contrast, we treat the training pool itself as a design variable, which to our knowledge has not been explored for LLM routing.*

Memory-based routing. Retrieval-augmented methods condition LLMs on similar past examples; a natural analog for routing is to retrieve past routing decisions that resemble the current state (Stripelis et al., 2024; Dann et al., 2025). *Our memory is case-based: it caches historical routing decisions and updates them as training proceeds.*

3 Preliminary

3.1 Agentic Systems and Execution Graphs

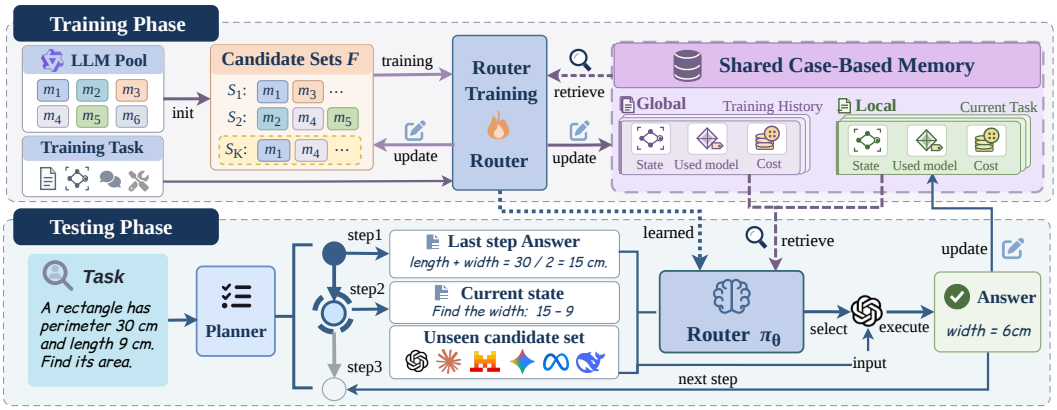


Figure 2: Overview of our method. During training, candidate LLM sets are constructed via a bi-level optimization framework: a candidate set selection score is derived, proven approximately submodular, and optimized with a greedy algorithm. The router is retrained on the augmented set. Meanwhile, a case-based memory module records routing decisions into local and global memory, and retrieved cases augment the router's state representation throughout training. During inference, the router selects an LLM at each node of the execution graph, augmented by the frozen global memory built during training along with a dynamically populated local memory.

Agentic systems typically produce an *execution trace* represented by a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for each input prompt, where each node $v \in \mathcal{V}$ is a step and each edge $(u, v) \in \mathcal{E}$ indicates that step v depends on the output of step u. Steps are executed in topological order. At step t, corresponding to node v_t, the model input x_t is an abstract representation of the *execution state*; its exact form depends on the specific execution model. For instance, in GraphPlanner Feng et al. (2026), x_t incorporates the execution history and the outputs of previous steps.

3.2 The General Routing Problem

At step t, the router takes the model input state x_t and selects a model from a *candidate set* $S \subseteq \mathcal{M}$, where $\mathcal{M}$ is the universe of available LLMs, according to a routing policy $\pi_{\boldsymbol{\theta}}(\cdot \mid x_t, S)$ parameterized by $\boldsymbol{\theta}$. The chosen model executes the step, and its output becomes part of the state for all subsequent steps. Routing therefore recurs at every node of the graph, and each decision shapes the context in which later decisions are made.

Algorithm 1 Greedy Candidate Set Construction

Require: Training pool $\mathcal{M}_{\text{train}}$, current training set $\mathscr{S}$, meta-gradient $\hat{\boldsymbol{g}}_{\text{meta}}$, stopping threshold $\tau > 0$
Ensure: Candidate set S
Initialize $S \leftarrow \emptyset$
repeat
  Compute marginal gain $\Delta(m|S) = f(S \cup \{m\} \mid \mathscr{S}) - f(S \mid \mathscr{S})$ for all $m \in \mathcal{M}_{\text{train}} \setminus S$
  $m^* \leftarrow \arg\max_m \Delta(m|S)$
  if $\Delta(m^*|S) < \tau$ **then**
    break ▷ marginal gain vanishes, stop adding
  $S \leftarrow S \cup \{m^*\}$
until $S = \mathcal{M}_{\text{train}}$ or $\Delta(m^*|S) < \tau$
return S

Formally, routing produces an assignment $\mathbf{a} = (a_1, a_2, \dots)$ with $a_t \in S$ at each step t, and its quality is measured by a task-level objective

$$J(\mathbf{a}) = U(\text{final answer}) - \lambda \sum\nolimits_t \text{Cost}(a_t), \tag{1}$$

where $U(\cdot) \in [0, 1]$ is the task utility, $\text{Cost}(a_t) \geq 0$ is the execution cost of the assigned LLM, and $\lambda > 0$ controls the trade-off between performance and cost.

The router is trained by minimizing a loss $\mathcal{L}(\boldsymbol{\theta})$ that operationalizes this objective. Our formulation is agnostic to the choice of $\mathcal{L}$: we assume only that $\mathcal{L}$ is differentiable in $\boldsymbol{\theta}$ and that its gradient can be estimated from sampled rollouts, labeled decisions, or paired comparisons. The same objective admits several instantiations. For instance, if reinforcement learning is used Feng et al. (2026; 2025), $J(\mathbf{a})$ serves directly as the reward and $\mathcal{L}(\boldsymbol{\theta})$ is the negative expected return, optimized by policy gradient. In supervised learning, an oracle supplies the best action at each step, and $\mathcal{L}(\boldsymbol{\theta})$ is the negative log-likelihood of that action under the policy, optimized by cross-entropy. We write $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ for the learned parameters.

Generalizability. In real-world deployments, the candidate set S is not fixed: it is determined by external factors such as model availability, licensing, or deployment constraints, and changes as new LLMs are released or old ones are deprecated (Wang et al., 2025). We therefore study routing models trained on candidate sets drawn from a training pool $\mathcal{M}$ that generalize to candidate sets drawn from a different pool $\mathcal{M}' \neq \mathcal{M}$, with the disjoint case $\mathcal{M}' \cap \mathcal{M} = \emptyset$ as the extreme setting and overlapping pools as a relaxation. Our formulation applies to any training loss $\mathcal{L}$ as defined above.

State-of-the-art routers address this by representing each LLM in S as an embedding e_i and scoring it against the input representation x_t, with the routing distribution given by a softmax over the candidate set (Jitkrittum et al., 2026; Wang et al., 2026; Feng et al., 2025). Since the score depends on a model only through its embedding, a model never seen during training can be scored once its embedding is computed by the same encoder, without retraining or observing its execution history (Jitkrittum et al., 2026; Wang et al., 2026; Feng et al., 2025).

4 METHOD

Figure 2 provides an overview of our method. It consists of two main components: a memory-augmented router architecture (§4.1) and a candidate LLM set construction procedure (§4.2). Illustrations of both components are provided in Appendix D.

4.1 ROUTER ARCHITECTURE

In this section, we present a case-based memory module for routing models in agentic systems, which aims to refer to similar cases in history and their optimal routing decisions so far to enhance the routing performance for current samples. This section comprises the construction of case-based memory in §4.1.1 and how to retrieve relevant case-based memory via attention for subsequent routing in §4.1.2.

4.1.1 CONSTRUCTING CASE-BASED MEMORY

Memory entries and encodings. The memory stores *cases*, each recording a completed routing decision at a step j, which consists of three fields: the execution state x_j, the model that was selected

Algorithm 2 Candidate LLM Set Construction and Router Training

Require: Training pool $\mathcal{M}_{\text{train}}$, validation pool $\mathcal{M}_{\text{val}}$, budget K (number of candidate sets), stopping threshold τ, initial policy parameters $\boldsymbol{\theta}$
Ensure: Trained router $\pi_{\boldsymbol{\theta}}$, training set $\mathscr{S}$
Initialize $\mathscr{S} \leftarrow \emptyset$, $\boldsymbol{\theta}$ randomly
for epoch $= 1$ to K **do**
  Phase 1: Compute meta-gradient
  Estimate $\hat{\boldsymbol{g}}_{\text{meta}}$ from PPO rollouts on $\mathcal{M}_{\text{val}}$
  Phase 2: Greedy construction of a new candidate set
  $S_{\text{new}} \leftarrow \textsc{GreedyCandidateSet}(\mathcal{M}_{\text{train}}, \mathscr{S}, \hat{\boldsymbol{g}}_{\text{meta}}, \tau)$ ▷ Algorithm 1
  $\mathscr{S} \leftarrow \mathscr{S} \cup \{S_{\text{new}}\}$
  Phase 3: Retrain router on the augmented training set
  for $i = 1$ to N_{rollouts} **do**
    Sample $S \sim \text{Uniform}(\mathscr{S})$
    Collect trajectories by running $\pi_{\boldsymbol{\theta}}$ on S
    Update $\boldsymbol{\theta}$ via the PPO objective
return $\boldsymbol{\theta}, \mathscr{S}$

for execution, e_j, and the observed cost of executing the selected LLM, κ_j. Formally, an entry in the case-based memory is

$$c_j = (x_j,\, e_j,\, \kappa_j)\,,$$

in which κ_j is normalized across all model costs, and we slightly abuse the notation by using x_j and e_j to denote the execution state representation and the model representation produced by the policy model $\boldsymbol{\theta}$. We keep these representations abstract so that *this memory module is compatible with arbitrary routing models that produce representations for each candidate LLM.*

Local and global case-based memory. We store two different sets of cases as memory, i.e., local memory, $\mathcal{M}_t^L$, and global memory, $\mathcal{M}_t^G$. The former one records all routing decisions of each step within one execution trace as individual cases. On the other hand, $\mathcal{M}_t^G$ consists of per-step cases constructed globally across all input queries.

Memory updates. A case c_j is written to $\mathcal{M}_t^L$ or $\mathcal{M}_t^G$ after the corresponding step is completed, which is not only appended to the local memory $\mathcal{M}_t^L$ during training, but also stored in the global base $\mathcal{M}^G$. If the execution state x_j in c_j is identical to an existing case, it replaces that case rather than being appended, so *each entry keeps the most updated routing decision for* x_j. In addition, the global memory, $\mathcal{M}_t^G$, operates under a fixed budget, evicting the oldest cases in first-in-first-out order once the budget is exceeded. During inference, the global memory is frozen while the local memory is dynamically populated for each input prompt.

4.1.2 Case Retrieval and Memory-augmented Routing

Attention-based case retrieval. For either local or global memory, $b \in \{L, G\}$, we maintain an attention module parameterized by $(W_Q^b, W_K^b, W_V^b, W_O^b)$ so that each execution state x_t and one candidate model e_i attends to its most similar memory case $c_j = (x_j, e_j^{\text{used}}, \kappa_j)$. Formally, we compute the query vector taking x_t and e_i as input, and the key and value vectors using x_j and e_j^{used}:

$$q_{t,i}^b = W_Q^b[x_t; e_i], \quad k_j^b = W_K^b[x_j; e_j^{\text{used}}], \quad v_j^b = W_V^b[x_j; e_j^{\text{used}}; \kappa_j], \tag{2}$$

which are used to compute the attention weights and output:

$$\alpha_{t,i,j}^b = \text{softmax}_{j \in \mathcal{B}_t^b}\left(\frac{q_{t,i}^b \cdot k_j^b}{\sqrt{d_b}}\right), \quad u_{t,i}^b = W_O^b \sum\nolimits_{j \in \mathcal{B}_t^b} \alpha_{t,i,j}^b v_j^b. \tag{3}$$

Intuitively, for each state x_t, a memory case c_j receives higher weight if its input prompt is highly similar to x_t and its stored decision matches the decision of choosing e_i for x_t. The resulting $u_{t,i}^b$ is the memory summary retrieved from base b for the pair (x_t, e_i), obtained by aggregating the values of the most relevant cases with their attention weights.

Memory-augmented state representation. The memory module augments the execution state representation x_t as follows:

$$\tilde{x}_{t,i} = x_t + \sum\nolimits_{b\in\{L,G\}} \psi(u^b_{t,i}), \tag{4}$$

where ψ projects the retrieved summary into the state representation space and bounds the projected summary to a fixed range, so the retrieved memory summary can shift x_t by a limited amount and never overwhelm the base router.

The augmented state $\tilde{x}_{t,i}$ can be substituted directly into any scores in the router model $\boldsymbol{\theta}$ that compare the state representation with the candidate model representation. In other words, the score $\mathrm{score}(x_t, e_i)$ is replaced with $\mathrm{score}(\tilde{x}_{t,i}, e_i)$. Meanwhile, all other modules remain the same as those in the base router. During training, the policy parameters $\boldsymbol{\theta}$ are then jointly trained with the above memory module parameters, $(W^b_Q, W^b_K, W^b_V, W^b_O, \psi)$, by minimizing the routing loss $\mathcal{L}(\boldsymbol{\theta})$ defined in §3.2.

Memory for different candidate LLM sets. Each case c_j is associated with the candidate LLM set S under which it was generated, so memory is indexed by candidate set. During inference, if the router encounters an unseen set S_{new}, we first identify the most similar training set $S^* \in \mathscr{S}$ and use its memory to augment routing on S_{new}. The similarity is measured by a score adapted from the candidate set selection score introduced in §4.2.2; see Appendix A for details.

4.2 Candidate LLM Set Selection

We select training candidate sets that provide complementary learning signals for routing with held-out LLM combinations. Our approach uses a bi-level formulation to motivate a local gradient-alignment objective, whose coverage structure enables efficient greedy selection.

4.2.1 Bi-Level Formulation

Let $\mathcal{M}_{\text{train}}$ and $\mathcal{M}_{\text{val}}$ be disjoint training and validation model pools. Let $\mathcal{C} \subseteq 2^{\mathcal{M}_{\text{train}}} \setminus \{\varnothing\}$ be a finite library of training candidate sets, from which we select a collection $\mathscr{S} \subseteq \mathcal{C}$. We write $\mathcal{L}(\boldsymbol{\theta}, S)$ for the routing loss in Section 3.2, making its candidate-set dependence explicit.

We construct nonempty held-out candidate sets $S'_1, \ldots, S'_R \subseteq \mathcal{M}_{\text{val}}$. For each S'_r, the loss $\mathcal{L}_{\text{val}}(\boldsymbol{\theta}, S'_r)$ is evaluated on the same validation-query distribution. For agentic routing, this loss includes complete executions with candidate models restricted to S'_r.

The ideal collection minimizes held-out loss after training:

$$\boldsymbol{\theta}^*(\mathscr{S}) \in \arg\min_{\boldsymbol{\theta}} \frac{1}{|\mathscr{S}|} \sum\nolimits_{S\in\mathscr{S}} \mathcal{L}(\boldsymbol{\theta}, S), \mathscr{S}^* \in \arg\min_{\mathscr{S}\subseteq\mathcal{C}, 1\le|\mathscr{S}|\le K} \sum\nolimits_{r=1}^{R} \mathcal{L}_{\text{val}}\big(\boldsymbol{\theta}^*(\mathscr{S}), S'_r\big). \tag{5}$$

Solving this problem directly requires repeated inner optimization. We instead select candidate sets using a local surrogate that rewards complementary transfer benefits across held-out model combinations.

4.2.2 Gradient-Alignment Coverage

Available training updates. Different training candidate sets can provide useful updates for different held-out combinations. Selecting sets independently by their aggregate usefulness may repeatedly favor the same combinations. We therefore evaluate a collection by the best local update it offers for each held-out candidate set.

At the current parameters $\boldsymbol{\theta}_0$, define the training and validation gradients:

$$\boldsymbol{g}_{\text{train}}(S) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_0, S), \quad \boldsymbol{g}_{\text{val}}(S'_r) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{val}}(\boldsymbol{\theta}_0, S'_r). \tag{6}$$

Each training set S supplies a hypothetical update $\boldsymbol{\theta}^+_S = \boldsymbol{\theta}_0 - \eta \boldsymbol{g}_{\text{train}}(S), \eta > 0$. A collection $\mathscr{S}$ thus offers the parameter choices $\mathcal{A}_\eta(\mathscr{S}) = \{\boldsymbol{\theta}_0\} \cup \{\boldsymbol{\theta}^+_S : S \in \mathscr{S}\}$, where $\boldsymbol{\theta}_0$ represents making no update. Its available held-out improvement is

$$V_\eta(\mathscr{S}) = \sum\nolimits_{r=1}^{R} \Big[\mathcal{L}_{\text{val}}(\boldsymbol{\theta}_0, S'_r) - \min\nolimits_{\boldsymbol{\vartheta}\in\mathcal{A}_\eta(\mathscr{S})} \mathcal{L}_{\text{val}}(\boldsymbol{\vartheta}, S'_r)\Big]. \tag{7}$$

This objective asks whether the collection contains an update that improves routing for each held-out model combination.

First-order selection score. Linearizing each validation loss motivates the positive alignment

$$w(S, S'_r) = \left[\boldsymbol{g}_{\text{val}}(S'_r)^\top \boldsymbol{g}_{\text{train}}(S)\right]_+, \tag{8}$$

where $[z]_+ = \max\{z, 0\}$. The positive part follows from the no-update option. Choosing the best available update for each S'_r then yields

$$F(\mathscr{S}) = \sum\nolimits_{r=1}^{R} \max\big(\{0\} \cup \{w(S, S'_r) : S \in \mathscr{S}\}\big). \tag{9}$$

Theorem 1 (First-order transfer coverage)**.** *Suppose the validation losses have β-Lipschitz gradients along the update segments and $\|\boldsymbol{g}_{\text{train}}(S)\|_2 \le G$ for all $S \in \mathcal{C}$. Then for every $\mathscr{S} \subseteq \mathcal{C}$:*

$$|V_\eta(\mathscr{S}) - \eta F(\mathscr{S})| \le \frac{R\beta\eta^2 G^2}{2}$$

Thus, F approximates the usefulness of the available training directions across held-out candidate sets. The maximum avoids repeatedly counting weaker directions serving the same held-out combination. This encourages diversity in transfer benefits, rather than merely diversity in model identities.

The available-update objective is a selection surrogate for equation 5: it allows different held-out sets to choose different updates and does not equal the effect of jointly retraining on $\mathscr{S}$. The proof is given in Appendix B.1.

4.2.3 Submodularity and Greedy Selection

During a selection phase, we hold $\boldsymbol{\theta}_0$, the held-out candidate sets, and all alignment scores fixed. Starting from $\mathscr{S}_0 = \varnothing$, we add the candidate set with the largest marginal gain:

$$S_t \in \arg\max_{S \in \mathcal{C} \setminus \mathscr{S}_t} f(S \mid \mathscr{S}_t), \quad f(S \mid \mathscr{S}) = F(\mathscr{S} \cup \{S\}) - F(\mathscr{S}), \tag{10}$$

and set $\mathscr{S}_{t+1} = \mathscr{S}_t \cup \{S_t\}$.

Theorem 2 (Submodular coverage)**.** *For fixed alignment scores, F is normalized, monotone, and submodular over $\mathcal{C}$. For $1 \le K \le |\mathcal{C}|$, K exact greedy additions yield*

$$F(\mathscr{S}_K) \ge (1 - 1/e) \max_{\mathscr{S} \subseteq \mathcal{C}, |\mathscr{S}| \le K} F(\mathscr{S}).$$

To evaluate marginal gains, maintain $h_r(\mathscr{S}) = \max\big(\{0\} \cup \{w(S, S'_r) : S \in \mathscr{S}\}\big)$. Then

$$f(S \mid \mathscr{S}) = \sum\nolimits_{r=1}^{R} [w(S, S'_r) - h_r(\mathscr{S})]_+.$$

The proof is in Appendix B.2. The submodularity enables a greedy construction of a candidate set that does not require a predefined pool size. Instead, we add models one at a time and stop when the marginal gain vanishes. This adaptive stopping criterion determines the pool size dynamically. This algorithm is outlined in Algorithm 1. Subsequently, we present the overall algorithm in Algorithm 2. At each round, we construct a new candidate set S_{new} by invoking Algorithm 1, add it to $\mathscr{S}$, and retrain the router on the augmented training set. This procedure dynamically builds a diverse training set while continuously adapting the router to its current weaknesses.

5 Experiments

5.1 Experimental setup

Datasets Following prior studies (Feng et al., 2026), we use five categories, each with a training dataset and a held-out dataset from the same skill area to test generalization: **Knowledge** (MMLU-Pro for training, GPQA-Diamond held out), **Math** (GSM8K and MATH for training, AIME 2024 held out), **Code** (MBPP for training, HumanEval held out), **Multi-hop QA** (HotpotQA-Distractor for training, MuSiQue-Ans held out), and **Commonsense & Logic** (CommonsenseQA for training, LogicGrid held out). The router is never trained on held-out examples, so improvements reflect genuine transfer across reasoning skills, providing a controlled testbed for generalizing to unseen LLMs and unseen candidate set combinations. Dataset descriptions, splits, and evaluation metrics are detailed in Appendix E.1.

LLM backbone The LLM candidate pools for training and validation, i.e., $\mathcal{M}_{\text{train}}$ and $\mathcal{M}_{\text{val}}$, consist of eight and three Qwen-series LLMs, respectively. The training candidates $\mathcal{M}_{\text{train}}$ are Qwen3-0.6B, 1.7B, 4B, 14B and Qwen3.5-2B, 4B, 9B, 27B. The held-out executors $\mathcal{M}_{\text{val}}$, used to assess meta-performance and to guide candidate pool and checkpoint selection, are Qwen3.5-0.8B, Qwen3-8B, and Qwen3-32B. For the final generalizability evaluation to unseen LLMs, we use GPT-5-full/mini/nano, GPT-4.1/full/mini/nano, together with Mistral Small/Medium/Large. Full model details are provided in Appendix E.2.

5.2 BASELINES

We compare against two groups of baselines. **Single-round invocation baselines** select one LLM for the entire query without multi-step decomposition: Router-KNN and Router-MLP (Shnitzer et al., 2024), Router-SVM (Hu et al., 2024), RouterDC (Chen et al., 2024b), GraphRouter (Feng et al., 2025), and ICL-Router (Wang et al., 2026). **Multi-round invocation baselines** route across multiple steps or rounds: Prompt LLM and Router-KNN-MR (Zhang et al., 2025), R2-Reasoner (Shao et al., 2026), Router-R1 (Zhang et al., 2025), and GraphPlanner (Feng et al., 2026).

To evaluate the effect of GREAT in an end-to-end manner, we incorporate GREAT into existing routing methods. For baselines that already contain a memory module, we replace it with ours. For baselines without a standalone memory module but whose architecture supports memory encodings, we augment them with both components. Baselines that do not support memory encodings are evaluated without incorporating GREAT. Details of adaptations are provided in Appendix F.1.

To isolate the effect of our candidate set selection mechanism, we compare against six alternative strategies, all selecting the same number of models as our method unless otherwise noted: 1) **Random**, which samples a candidate set of matched size uniformly from $\mathcal{M}_{\text{train}}$; 2) **Singleton score**, which ranks models by their individual score and selects the top-K in one shot; 3) **coverage**, which maximizes coverage of the model space via a greedy procedure (Nemhauser et al., 1978); 4) **Diversity-aware**, which combines the singleton score with a diversity penalty balanced by a hyperparameter (Carbonell & Goldstein, 1998); 5) **Determinantal point process**, which enforces diversity through a determinant-based marginal gain (Chen et al., 2018); 6) **Maximum mean discrepancy**, which encourages the selected set to match the distribution of the full model pool (Chen et al., 2010). The mathematical definitions of each strategy are provided in Appendix C.1.

5.3 EXPERIMENTAL RESULTS

When $\mathcal{M}_{\text{train}} \cap \mathcal{M}_{\text{test}} = \emptyset$, GREAT improves average accuracy by **1.6–7.1 pp** and reduces average cost by **8.8%–36.5%** across all five backbones (Table 1). *The largest gains appear for ICL-Router and Router-R1, the smallest for RouterDC.* Even the strongest baseline, GraphPlanner, improves in both accuracy and cost when augmented with GREAT, and *the pattern is consistent across all five categories.* To further comprehensively evaluate the accuracy–cost trade-off, we vary λ in equation 1 and plot the Pareto curve in Figure 8 (Appendix G.10), which shows that incorporating GREAT into baseline methods generally yields better accuracy–cost trade-offs.

In addition, we report the results of GREAT and baseline methods when $\mathcal{M}_{\text{train}} = \mathcal{M}_{\text{test}}$ in Table 10 in Appendix G.1, which indicate consistent performance gains of GREAT over baselines.

The candidate set selection mechanism is the main driver of these gains. With GraphPlanner and case-based memory fixed across strategies (*No* disables only set selection), our strategy achieves both the highest average accuracy and the lowest average execution cost (Table 2). Compared against six alternative strategies, GREAT achieves **87.0%** average accuracy at cost **59.4**, outperforming the best baseline by **3.0 pp** with **35% less cost**. *Random selection and the no-selection variant fall far behind, confirming that the improvement comes from principled construction rather than simply adding more models.*

These improvements also hold on the five held-out datasets (Table 3): **every backbone improves in both accuracy and cost** when augmented with GREAT, with average accuracy gains of **3.3–5.2 pp** and cost reductions of **8.2–57.6%**.

5.4 ABLATION STUDIES

Effect of case-based memory. Figure 3(a) shows that removing either local or global memory reduces accuracy and increases cost, with global memory having the larger and more consistent impact across categories. *The two memories are complementary:* local captures within-trace dependencies, while global transfers cross-query routing patterns.

Transductive vs. inductive memory. Table 4 shows that retaining training-time global memory during inference time (transductive) improves accuracy by 0.4 pp and reduces cost by 6.1% for GraphPlanner+GREAT compared to excluding it (inductive). *This confirms that global memory encodes transferable routing knowledge useful even when test pools differ from training.*

Table 1: Routing with unseen LLMs. Bold marks the better value within each backbone pair (higher Acc, lower Cost); ties highlight only +GREAT. Δ Avg. gives changes from each backbone to +GREAT: accuracy in percentage points (pp) and relative cost in %.

Setting	Knowledge Acc	Knowledge Cost	Math Acc	Math Cost	Code Acc	Code Cost	Multi-hop QA Acc	Multi-hop QA Cost	Common. & Logic Acc	Common. & Logic Cost	Average Acc	Average Cost	Δ Avg. Δ Acc (pp)	Δ Avg. Δ Cost (%)
Single-round Router														
RouterDC	75.0%	30.7	78.0%	50.4	74.0%	**13.2**	83.0%	29.1	84.0%	16.3	78.8%	28.0	–	–
RouterDC+GREAT	**77.0%**	**24.8**	**80.0%**	**46.7**	**76.0%**	13.3	**83.0%**	**27.1**	**86.0%**	**15.5**	**80.4%**	**25.5**	**+1.6**	**-8.8%**
GraphRouter	75.5%	21.8	73.0%	30.6	76.0%	32.9	82.8%	33.0	84.0%	51.7	78.3%	34.0	–	–
GraphRouter+GREAT	**78.0%**	**19.9**	**82.0%**	**27.6**	**78.0%**	**10.1**	**84.4%**	**25.8**	**84.0%**	**39.1**	**81.3%**	**24.5**	**+3.0**	**-27.9%**
ICL-Router	70.0%	72.0	61.0%	48.4	74.0%	38.4	85.2%	75.9	74.0%	25.7	72.8%	52.1	–	–
ICL-Router+GREAT	**78.0%**	**44.8**	**82.0%**	**41.5**	**74.0%**	**28.4**	**87.7%**	**40.2**	**78.0%**	**10.6**	**79.9%**	**33.1**	**+7.1**	**-36.5%**
Multi-round Router														
Router-R1	20.0%	19.9	24.0%	**7.3**	38.0%	16.0	21.0%	56.8	60.0%	17.5	32.6%	23.5	–	–
Router-R1+GREAT	**30.0%**	**13.8**	**33.0%**	7.8	**38.0%**	**16.0**	**25.0%**	**20.3**	**64.0%**	**17.1**	**38.0%**	**15.0**	**+5.4**	**-36.2%**
GraphPlanner	80.0%	123.0	90.0%	58.9	84.0%	82.8	88.4%	64.9	78.0%	28.2	84.1%	71.6	–	–
GraphPlanner+GREAT	**84.0%**	**113.3**	**92.0%**	**51.6**	**84.0%**	**73.9**	**89.1%**	**55.5**	**80.0%**	**27.6**	**85.8%**	**64.4**	**+1.7**	**-10.1%**

Table 2: Candidate Set selection.

Strategy	Average Acc.	Average Cost
No	76.2	59.6
Random	76.7	70.5
Singleton	83.2	95.8
Sim-Coverage	81.9	86.6
Sim-diversity	84.0	91.6
DPP	83.2	95.8
MMD	83.9	81.9
GREAT	**87.0**	**59.4**

Table 3: Unseen LLMs and Unseen datasets.

Router	Avg. Acc. Base	Avg. Acc. +GREAT	Avg. Cost Base	Avg. Cost +GREAT
RouterDC	63.0	**67.0**	38.3	**26.5**
GraphRouter	64.4	**68.5**	54.1	**40.6**
ICL-Router	64.1	**68.3**	52.4	**22.2**
Router-R1	25.4	**30.6**	69.2	**55.5**
GraphPlanner	68.8	**72.1**	140.2	**128.7**

Table 4: Memory reuse.

Mode	Average Acc.	Average Cost
GraphPlanner		
Ind.	77.4	31.9
Trans.	77.4	**30.1**
GraphPlanner+GREAT		
Ind.	82.8	97.4
Trans.	**83.2**	**91.5**

Five-category averages; Acc. (%) ↑, Cost ↓. Full results: Appendices G.6, G.7, and G.8 (left to right).

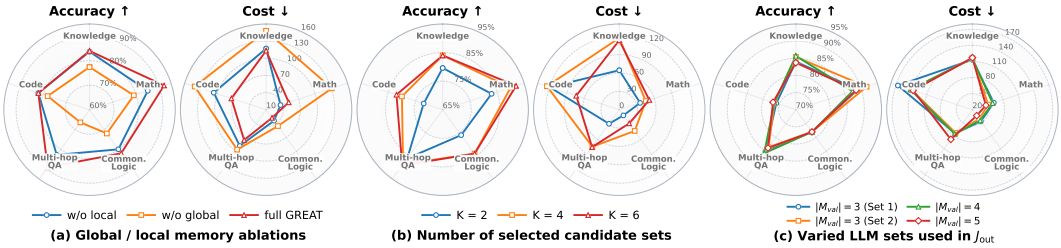


Figure 3: Ablation studies with GraphPlanner+GREAT: (a) local and global memory; (b) the number of training candidate sets; (c) validation LLM pools used in J_{out}. Each panel reports accuracy (higher is better) and execution cost (lower is better) across five task categories.

Number of selected candidate sets. Figure 3(b) varies the number of candidate sets K. Increasing K from 2 to 4 yields consistent gains, while increasing from 4 to 6 brings only marginal improvement, indicating that a small number of diverse sets suffices. Based on this observation, we configure $K = 4$ by default.

Robustness to validation pool composition. Figure 3(c) evaluates different validation pools $\mathcal{M}_{\text{val}}$ by varying its size as well as specific sets. *Performance is stable across all configurations,* showing that GREAT does not depend on a particular choice of $\mathcal{M}_{\text{val}}$.

Effect of LLM sets used in J_{out} Figure 3(c) shows that average routing accuracy remains relatively stable across the tested compositions and sizes of $\mathcal{M}_{\text{val}}$. Detailed numerical results are provided in Appendix G.4. Other ablation studies, including the effect of the global memory size, and training efficiency analysis are delayed to Appendix G.

6 Conclusion

We introduced a memory-augmented router that enhances generalizability to unseen LLMs in agentic systems. Our approach combines a case-based memory that caches and retrieves past routing decisions with a candidate set selection score derived from bi-level optimization and optimized via a submodularity-based greedy algorithm. Experiments across five reasoning domains show improved transfer to unseen LLMs and candidate combinations over single-round and multi-round routing baselines.

AI USE STATEMENT

Generative AI tools were used to assist literature discovery, method and experiment-design feedback, implementation and debugging, experiment monitoring, result organization, scientific-figure drafting, and manuscript outlining and language revision. The authors determined the research questions and experimental decisions, inspected the source code and raw artifacts, verified numerical claims and citations against primary sources, and take responsibility for the paper in full.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. AutoMix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 131000–131034, 2024. doi: 10.52202/079017-4164. URL https://doi.org/10.52202/079017-4164.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pp. 335–336, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130155. doi: 10.1145/290941.291025. URL https://doi.org/10.1145/290941.291025.

Laming Chen, Guoxin Zhang, and Eric Zhou. Fast greedy MAP inference for determinantal point process to improve recommendation diversity. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/dbbf603ff0e99629dda5d75b6f75f966-Abstract.html.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=cSimKw5p6R.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 66305–66328, 2024b. doi: 10.52202/079017-2120. URL https://doi.org/10.52202/079017-2120.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 109–116, 2010. URL https://dl.acm.org/doi/10.5555/3023549.3023562.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Christoph Dann, Yishay Mansour, Teodor Vanislavov Marinov, and Mehryar Mohri. Principled model routing for unknown mixtures of source domains. In *Advances in Neural Information Processing Systems*, volume 38, pp. 40626–40657, 2025. doi: 10.52202/085713-1356. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/39c60dda48ebf0a2e5dda52ce08eb5c8-Abstract-Conference.html.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/b47d93c99fa22ac0b377578af0a1f63a-Abstract-Conference.html.

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. BEST-route: Adaptive LLM routing with test-time optimal compute. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 13870–13884. PMLR, 2025. URL https://proceedings.mlr.press/v267/ding25d.html.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for LLM selections. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=eU39PDsZtT.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. GraphPlanner: Graph memory-augmented agentic routing for multi-agent LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/c86ed90b14e55f2ecf838a755e404b06-Abstract-Conference.html.

Neel Guha, Mayee Chen, Trevor Chow, Ishan Khare, and Christopher Ré. Smoothie: Label-free language model routing. In *Advances in Neural Information Processing Systems*, volume 37, pp. 127645–127672, 2024. doi: 10.52202/079017-4053. URL https://doi.org/10.52202/079017-4053.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran Associates, Inc., 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

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.

Zhongzhan Huang, Guoming Ling, Yupei Lin, Yandong Chen, Shanshan Zhong, Hefeng Wu, and Liang Lin. RouterEval: A comprehensive benchmark for routing LLMs to explore model-level scaling up in LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 3860–3887. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.208. URL https://aclanthology.org/2025.findings-emnlp.208/.

Ruihan Jin, Pengpeng Shao, Zhengqi Wen, Jinyang Wu, Mingkuan Feng, Shuai Zhang, and Jianhua Tao. RadialRouter: Structured representation for efficient and robust large language models routing. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 14587–14600. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.787. URL https://aclanthology.org/2025.findings-emnlp.787/.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon,

and Sanjiv Kumar. Universal model routing for efficient LLM inference. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ka82fvJ5f1.

Mathematical Association of America. American invitational mathematics examination (AIME). MAA Invitational Competitions, n.d. URL https://maa.org/maa-invitational-competitions/. Accessed September 26, 2026.

Mistral AI. Introducing Mistral 3. Mistral AI, December 2025a. URL https://mistral.ai/news/mistral-3/.

Mistral AI. Medium is the new large. Mistral AI, May 2025b. URL https://mistral.ai/news/mistral-medium-3/.

Mistral AI. Mistral Small 3.1. Mistral AI, March 2025c. URL https://mistral.ai/news/mistral-small-3-1/.

Arindam Mitra and Chitta Baral. Learning to automatically solve logic grid puzzles. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1023–1033. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1118. URL https://aclanthology.org/D15-1118/.

Yasmin Moslem and John Kelleher. Dynamic model routing and cascading for efficient LLM inference: A survey. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=ypRg1TvQaM. Survey Certification.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978. doi: 10.1007/BF01588971. URL https://link.springer.com/article/10.1007/BF01588971.

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 from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

OpenAI. Introducing GPT-4.1 in the API. OpenAI, April 2025a. URL https://openai.com/index/gpt-4-1/.

OpenAI. Introducing GPT-5 for developers. OpenAI, August 2025b. URL https://openai.com/index/introducing-gpt-5-for-developers/.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive LLM routing under budget constraints. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23934–23949. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.1301. URL https://aclanthology.org/2025.findings-emnlp.1301/.

Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. xRouter: Training cost-aware LLMs orchestration system via reinforcement learning, 2025. URL https://arxiv.org/abs/2510.08439.

Qwen Team. Qwen3.5: Towards native multimodal agents. Qwen technical blog, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Soham Shah and Kumar Shridhar. Select-then-route: Taxonomy guided routing for LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 425–441. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-industry.28. URL https://aclanthology.org/2025.emnlp-industry.28/.

Chenyang Shao, Xinyang Liu, Yutang Lin, Fengli Xu, and Yong Li. Route-and-reason: Energy-efficient scaling of LLM reasoning via reinforced model routing. In *Proceedings of the ACM Web Conference 2026*. Association for Computing Machinery, 2026. doi: 10.1145/3774904.3793038. URL https://doi.org/10.1145/3774904.3793038.

Yuanzhe Shen, Yide Liu, Zisu Huang, Ruicheng Yin, Xiaoqing Zheng, and Xuanjing Huang. SATER: A self-aware and token-efficient approach to routing and cascading. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10515–10529. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.531. URL https://aclanthology.org/2025.emnlp-main.531/.

Reza Shirkavand, Shangqian Gao, Peiran Yu, and Heng Huang. Cost-aware contrastive routing for LLMs. In *Advances in Neural Information Processing Systems*, volume 38, pp. 155627–155659, 2025. doi: 10.52202/085713-5202. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/e46eb6403af68506331f941282d838aa-Abstract-Conference.html.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Zb0ajZ7vAt.

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 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15629–15644. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.761. URL https://aclanthology.org/2025.acl-long.761/.

Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. TensorOpera Router: A multi-model router for efficient LLM inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 452–462. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-industry.34. URL https://aclanthology.org/2024.emnlp-industry.34/.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421/.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Asterios Tsiourvas, Wei Sun, and Georgia Perakis. Causal LLM routing: End-to-end regret minimization from observational data. In *Advances in Neural Information Processing Systems*, volume 38, pp. 37289–37314, 2025. doi: 10.52202/085713-1252. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/357774d53e5ee21c5f08ba779e3b5dd9-Abstract-Conference.html.

Clovis Varangot-Reille, Christophe Bouvard, Mathieu Ciancone, Antoine Gourru, Marion Schaeffer, and François Jacquenet. Doing more with less: A survey on routing strategies for resource optimisation in large language model-based systems. *Journal of Artificial Intelligence Research*, 86, 2026a. doi: 10.1613/jair.1.19801. URL https://www.jair.org/index.php/jair/article/view/19801.

Clovis Varangot-Reille, Christophe Bouvard, and Antoine Gourru. Generalising LLM routing using past performance retrieval: A few-shot router is sufficient. In Selene Baez Santamaria, Sai Ashish Somayajula, and Atsuki Yamaguchi (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 304–319, Rabat, Morocco, March 2026b. Association for Computational Linguistics. ISBN 979-8-89176-383-8. doi: 10.18653/v1/2026.eacl-srw.22. URL https://aclanthology.org/2026.eacl-srw.22/.

Chenxu Wang, Hao Li, Yiqun Zhang, Linyao Chen, Jianhao Chen, Ping Jian, Qiaosheng Zhang, and Shuyue Hu. ICL-Router: In-context learned model representations for LLM routing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(39):33413–33421, 2026. doi: 10.1609/aaai.v40i39.40628. URL https://ojs.aaai.org/index.php/AAAI/article/view/40628.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. MixLLM: Dynamic routing in mixed large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10912–10922. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.545. URL https://aclanthology.org/2025.naacl-long.545/.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Fangzhou Wu and Sandeep Silwal. Efficient training-free online routing for high-volume multi-LLM serving. In *Advances in Neural Information Processing Systems*, volume 38, pp. 137112–137152, 2025. doi: 10.52202/085713-4577. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/c878e34a1b2095ae8177961ba24d642d-Abstract-Conference.html.

Jingjun Xu, Hongji Pu, Tao Feng, Haozhen Zhang, Jiaxuan You, and Ge Liu. RouteProfile: Graph-based profiling for cold-start LLM routing, 2026. URL https://arxiv.org/abs/2605.00180.

Cheng Yan, Wuyang Zhang, Zhiyuan Ning, Fan Xu, Ziyang Tao, Lu Zhang, Bing Yin, and Yanyong Zhang. Breaking model lock-in: Cost-efficient zero-shot LLM routing via a universal latent space. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(43):36483–36490, 3 2026. doi: 10.1609/aaai.v40i43.40970. URL https://ojs.aaai.org/index.php/AAAI/article/view/40970.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Jiahao Zeng, Ming Tang, and Ningning Ding. Learning to route llms from implicit cost-performance preferences via meta-learning. In Diego Calvanese (ed.), *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-26*, pp. 6045–6053. International Joint Conferences on Artificial Intelligence Organization, 8 2026. doi: 10.24963/ijcai.2026/673. URL https://doi.org/10.24963/ijcai.2026/673. Main Track.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-R1: Teaching LLMs multi-round routing and aggregation via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 156646–156678, 2025. doi: 10.52202/085713-4719. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/ceaa137fce916aba5c65fceb1309088b-Abstract-Conference.html.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning compact representations of large language models. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/bf5e4b85d203481d6e37bd32d9600162-Abstract-Conference.html.

A Memory Retrieval for Unseen Candidate Sets

During inference, the router may encounter an unseen candidate set S_{new}. Since memory is indexed by candidate set, we identify the most similar training set $S^* \in \mathscr{S}$ and use its memory to augment routing on S_{new}.

We adapt the candidate set selection score to measure set-to-set similarity by replacing the meta-gradient with the training gradient of one set:

$$\text{sim}(S_1, S_2) = \hat{\boldsymbol{g}}_{\text{train}}(S_1)^\top \hat{\boldsymbol{g}}_{\text{train}}(S_2). \tag{11}$$

This score is large when two sets induce similar training gradients, meaning that training on one would also improve performance on the other. Given S_{new}, we retrieve

$$S^* = \arg\max_{S \in \mathscr{S}} \hat{\boldsymbol{g}}_{\text{train}}(S)^\top \hat{\boldsymbol{g}}_{\text{train}}(S_{\text{new}}), \tag{12}$$

and use the memory associated with S^* to augment the router's state representation for S_{new}, following Section 4.1.2.

The gradients $\hat{\boldsymbol{g}}_{\text{train}}(S)$ for $S \in \mathscr{S}$ are precomputed after training and stored alongside the memory. At inference, computing $\hat{\boldsymbol{g}}_{\text{train}}(S_{\text{new}})$ requires a single gradient evaluation, after which retrieval reduces to a dot product against the stored gradients.

B Theoretical analysis

B.1 Proof of Theorem 1

Local adaptation surrogate. Equation 5 evaluates the router after joint optimization over a selected collection. We instead score the availability of useful local training directions.

For a nonempty collection $\mathscr{S}$, consider the scenario-wise local adaptation objective

$$\widetilde{J}_{\text{out}}(\mathscr{S}) = \sum_{r=1}^{R} \min_{\substack{S \in \mathscr{S} \\ \alpha \in [0,1]}} \mathcal{L}_r^{\text{val}} \left(\boldsymbol{\theta}_0 - \eta \alpha \boldsymbol{g}_{\text{train}}(S) \right).$$

Each scenario can choose a candidate training direction and a step fraction, including no update. Its improvement over the current router is

$$\sum_{r=1}^{R} \mathcal{L}_r^{\text{val}}(\boldsymbol{\theta}_0) - \widetilde{J}_{\text{out}}(\mathscr{S}) = \mathcal{I}_\eta(\mathscr{S}).$$

This scenario-wise adaptation objective motivates coverage. It is not identical to the outer objective after joint retraining in equation 5.

Proof. Fix a validation scenario r, candidate set S, and $\alpha \in [0, 1]$. Abbreviate

$$\boldsymbol{g}_S = \boldsymbol{g}_{\text{train}}(S), \qquad \boldsymbol{g}_r = \boldsymbol{g}_{\text{val},r}.$$

Assume that $\mathcal{L}_r^{\text{val}}$ has a β-Lipschitz gradient on a neighborhood containing the segment $\{\boldsymbol{\theta}_0 - \eta\alpha \boldsymbol{g}_S : \alpha \in [0, 1]\}$. The smoothness remainder bound gives

$$\mathcal{L}_r^{\text{val}}(\boldsymbol{\theta}_0 - \eta\alpha \boldsymbol{g}_S) = \mathcal{L}_r^{\text{val}}(\boldsymbol{\theta}_0) - \eta\alpha \boldsymbol{g}_r^\top \boldsymbol{g}_S + R_{r,S}(\alpha),$$

where

$$|R_{r,S}(\alpha)| \le \frac{\beta\eta^2\alpha^2}{2} \|\boldsymbol{g}_S\|_2^2 \le \frac{\beta\eta^2}{2} \|\boldsymbol{g}_S\|_2^2.$$

Define the improvement at step fraction α by

$$D_{r,S}(\alpha) = \mathcal{L}_r^{\text{val}}(\boldsymbol{\theta}_0) - \mathcal{L}_r^{\text{val}}(\boldsymbol{\theta}_0 - \eta\alpha \boldsymbol{g}_S).$$

Uniformly over $\alpha \in [0, 1]$,

$$\left| D_{r,S}(\alpha) - \eta\alpha \boldsymbol{g}_r^\top \boldsymbol{g}_S \right| \le \frac{\beta\eta^2}{2} \|\boldsymbol{g}_S\|_2^2.$$

For two functions on the same domain, the absolute difference of their maxima is bounded by their maximum absolute difference. Therefore,

$$\left| \max_{\alpha\in[0,1]} D_{r,S}(\alpha) - \max_{\alpha\in[0,1]} \eta\alpha \boldsymbol{g}_r^\top \boldsymbol{g}_S \right| \le \frac{\beta\eta^2}{2}\|\boldsymbol{g}_S\|_2^2.$$

Since $\eta > 0$,

$$\max_{\alpha\in[0,1]} \eta\alpha \boldsymbol{g}_r^\top \boldsymbol{g}_S = \eta[\boldsymbol{g}_r^\top \boldsymbol{g}_S]_+ = \eta w_{r,S}.$$

Thus,

$$|I_r(S) - \eta w_{r,S}| \le \frac{\beta\eta^2}{2}\|\boldsymbol{g}_S\|_2^2 \le E, \qquad E := \frac{\beta\eta^2 G^2}{2}. \tag{13}$$

Apply the same maximum-stability inequality across the selected candidate sets, including the common zero option:

$$\left|\max\big(\{0\} \cup \{I_r(S) : S \in \mathscr{S}\}\big) - \eta \max\big(\{0\} \cup \{w_{r,S} : S \in \mathscr{S}\}\big)\right| \le E.$$

Summing over the R validation scenarios yields

$$|\mathcal{I}_\eta(\mathscr{S}) - \eta F(\mathscr{S})| \le RE = \frac{R\beta\eta^2 G^2}{2}.$$

For $\mathscr{S} = \varnothing$, both objectives equal zero, so the result also holds. □

B.2 Proof of Theorem 2

Proof. All gradients and hence all weights $w_{r,S}$ are fixed throughout the proof.

Normalization, monotonicity, and submodularity. For each scenario r, define

$$h_r(\mathscr{S}) = \max\big(\{0\} \cup \{w_{r,S} : S \in \mathscr{S}\}\big).$$

Clearly, $h_r(\varnothing) = 0$ and $h_r \ge 0$. For $\mathscr{A} \subseteq \mathscr{B}$,

$$h_r(\mathscr{A}) \le h_r(\mathscr{B}),$$

so h_r is monotone.

For $S \notin \mathscr{S}$, its marginal gain is

$$h_r(\mathscr{S} \cup \{S\}) - h_r(\mathscr{S}) = [w_{r,S} - h_r(\mathscr{S})]_+.$$

Thus, if $\mathscr{A} \subseteq \mathscr{B}$ and $S \notin \mathscr{B}$,

$$[w_{r,S} - h_r(\mathscr{A})]_+ \ge [w_{r,S} - h_r(\mathscr{B})]_+.$$

This is the diminishing-returns property. Therefore, each h_r is submodular, and so is

$$F(\mathscr{S}) = \sum_{r=1}^{R} h_r(\mathscr{S}).$$

Normalization, nonnegativity, and monotonicity are also preserved under this sum.

Greedy guarantee. Let

$$\mathscr{S}^\star \in \arg\max_{\substack{\mathscr{S}\subseteq\mathcal{C}\\ |\mathscr{S}|\le K}} F(\mathscr{S}),$$

and let $\mathscr{S}_t$ be the collection after t greedy additions, starting from $\mathscr{S}_0 = \varnothing$.

For $t < K$, monotonicity and submodularity imply

$$\begin{aligned} F(\mathscr{S}^\star) - F(\mathscr{S}_t) &\le F(\mathscr{S}^\star \cup \mathscr{S}_t) - F(\mathscr{S}_t) \\ &\le \sum_{S\in\mathscr{S}^\star\setminus\mathscr{S}_t} [F(\mathscr{S}_t \cup \{S\}) - F(\mathscr{S}_t)] \\ &\le K \max_{S\in\mathcal{C}\setminus\mathscr{S}_t} [F(\mathscr{S}_t \cup \{S\}) - F(\mathscr{S}_t)]. \end{aligned}$$

By the greedy selection rule,

$$F(\mathscr{S}_{t+1}) - F(\mathscr{S}_t) \geq \frac{F(\mathscr{S}^\star) - F(\mathscr{S}_t)}{K}.$$

Writing

$$D_t = F(\mathscr{S}^\star) - F(\mathscr{S}_t),$$

we obtain

$$D_{t+1} \leq \left(1 - \frac{1}{K}\right) D_t.$$

Because $F(\mathscr{S}_0) = 0$, unrolling gives

$$D_K \leq \left(1 - \frac{1}{K}\right)^K F(\mathscr{S}^\star).$$

Hence,

$$F(\mathscr{S}_K) \geq \left[1 - \left(1 - \frac{1}{K}\right)^K\right] F(\mathscr{S}^\star) \geq (1 - 1/e) F(\mathscr{S}^\star).$$

□

B.3 Applicability and Implementation

Multi-step agentic routing. The update in Theorem 1 is a parameter-optimization step, not an execution step. For reinforcement learning, the loss may be

$$\mathcal{L}(\boldsymbol{\theta}, S) = -\mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}(\cdot|S)}[J(\mathbf{a}(\tau))],$$

where τ denotes a complete execution trace and $\mathbf{a}(\tau)$ its routing assignment.

If executor transitions are independent of $\boldsymbol{\theta}$ and the usual differentiation and integrability conditions hold, the policy gradient is

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, S) = -\mathbb{E}_\tau \left[J(\mathbf{a}(\tau)) \sum_{t=1}^{T(\tau)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid x_t, S) \right].$$

The state x_t incorporates the available execution history, so this gradient accounts for the effects of earlier routing decisions on later states and the final return. The coverage analysis therefore does not assume independent execution steps. It also applies to differentiable supervised or comparison-based routing losses.

When the router uses global memory, that memory bank is held fixed for the local analysis and throughout a selection phase. Within-trace local memory updates remain part of the execution dynamics.

Gradient estimates. Theorem 1 concerns exact gradients of the stated losses. In practice, we use estimates $\hat{\boldsymbol{g}}_{\mathrm{train}}(S)$ and $\hat{\boldsymbol{g}}_{\mathrm{val},r}$ and construct

$$\hat{w}_{r,S} = \left[\hat{\boldsymbol{g}}_{\mathrm{val},r}^\top \hat{\boldsymbol{g}}_{\mathrm{train}}(S)\right]_+.$$

For any fixed realized estimates, the resulting coverage function $\hat{F}$ remains normalized, monotone, and submodular. Consequently, the greedy guarantee applies to $\hat{F}$.

A guarantee relative to exact-gradient coverage requires an additional estimation bound. For example, if

$$|\hat{w}_{r,S} - w_{r,S}| \leq \zeta \qquad \text{for all } r, S,$$

then maximum stability gives

$$|\hat{F}(\mathscr{S}) - F(\mathscr{S})| \leq R\zeta.$$

Let $\hat{\mathscr{S}}_K$ be greedy for $\hat{F}$ and $\mathscr{S}^\star$ be optimal for F under budget K. Then

$$F(\hat{\mathscr{S}}_K) \geq (1 - 1/e) F(\mathscr{S}^\star) - (2 - 1/e) R\zeta.$$

This follows by transferring the greedy guarantee from $\hat{F}$ to F using the uniform error bound. The bound on ζ must be justified separately.

If a backbone supplies a surrogate-loss gradient, the local interpretation applies to that surrogate loss; it does not automatically imply the same improvement in expected task utility.

Library construction and computational cost. The ground set of the submodular optimization is the library $\mathcal{C}$ of candidate LLM sets. The library can contain all admissible subsets for small training pools, or a sampled collection for larger pools. Exact enumeration remains exponential in $|\mathcal{M}_{\text{train}}|$; submodularity makes collection selection efficient given the library, not library generation itself.

Let $N = |\mathcal{C}|$ and let d be the gradient dimension. Once gradients are available, forming all alignment weights costs $O(RNd)$. Maintaining the scenario-wise maxima h_r allows naive greedy selection in $O(KRN)$ additional time. Gradient estimation and executor rollout costs are separate from these arithmetic costs.

Stopping and gradient refresh. The cardinality guarantee assumes K greedy additions with fixed weights. If construction stops after $k \geq 1$ additions, the returned collection is a greedy prefix and retains the same approximation factor relative to the best collection of size at most k.

If stopping occurs because every remaining marginal gain is at most τ, then for any collection $\mathscr{T}$ with $|\mathscr{T}| \leq K$,

$$F(\mathscr{T}) \leq F(\mathscr{S}_k) + K\tau.$$

Indeed, monotonicity and submodularity give

$$F(\mathscr{T}) - F(\mathscr{S}_k) \leq \sum_{S \in \mathscr{T} \setminus \mathscr{S}_k} \Delta(S \mid \mathscr{S}_k) \leq K\tau.$$

Refreshing gradients changes the coverage function. Accordingly, the guarantee applies separately to each phase with frozen gradients, not to an entire sequence of interleaved selection and training updates.

Scope of the generalization claim. Disjoint validation models provide a transfer-oriented selection signal. Theorems 1 and 2 establish local update-availability and coverage guarantees. They do not guarantee the outcome of joint retraining or performance on an arbitrary unseen test-model distribution.

C Additional experimental setups

C.1 Candidate Set Selection Baselines

We provide the mathematical definitions of the candidate set selection baselines listed in Section 5. Throughout, $\mathcal{M}$ denotes the full model pool, $\mathscr{S}$ the collection of candidate sets used for training, $f(\cdot \mid \mathscr{S})$ the candidate set selection score defined in equation 10, $\text{sim}(\cdot, \cdot)$ a similarity function between model representations, and $\Delta(m \mid S) = f(S \cup \{m\} \mid \mathscr{S}) - f(S \mid \mathscr{S})$ the marginal gain of adding m to S.

Random. A candidate set S of matched size is sampled uniformly from $\mathcal{M}_{\text{train}}$, with no scoring function.

Singleton score. Models are ranked by their individual score

$$\Delta(m \mid \mathscr{S}) = f(\{m\} \mid \mathscr{S}),$$

and the top-K models are selected in one shot. No greedy procedure is used.

Facility / coverage. The set function is

$$f(S) = \sum_{i \in \mathcal{M}} \max_{j \in S} \text{sim}(i, j),$$

and the set is constructed greedily using

$$\Delta(m \mid S) = f(S \cup \{m\}) - f(S),$$

following the classical greedy algorithm for submodular maximization (Nemhauser et al., 1978).

Diversity-aware. The marginal gain combines the singleton score with a diversity penalty:

$$\Delta(m \mid S) = \lambda\, f(\{m\} \mid \mathscr{S}) - (1-\lambda) \max_{s \in S} \mathrm{sim}(s, m),$$

where $\lambda \in [0, 1]$ balances relevance and diversity (Carbonell & Goldstein, 1998).

Determinantal point process. Define the kernel matrix

$$L(i, j) = f(\{i\} \mid \mathscr{S}) \cdot \mathrm{sim}(i, j) \cdot f(\{j\} \mid \mathscr{S}),$$

where $L(S)$ denotes the submatrix indexed by S. The marginal gain is

$$\Delta(m \mid S) = \log\det\big(L(S \cup \{m\})\big) - \log\det\big(L(S)\big),$$

following the fast greedy MAP inference for determinantal point processes (Chen et al., 2018).

Maximum mean discrepancy. Let $\mu_S = \frac{1}{|S|}\sum_{m \in S} \phi(m)$ denote the mean feature of the selected set, where $\phi(\cdot)$ is the model representation. The discrepancy between the selected set and the remaining pool is

$$\mathrm{MMD}(S, V) = \|\mu_S - \mu_V\|_2, \qquad V = \mathcal{M} \setminus \bigcup \mathscr{S}.$$

The marginal gain is

$$\Delta(m \mid S) = \mathrm{MMD}^2(S, V) - \mathrm{MMD}^2(S \cup \{m\}, V),$$

which selects models that reduce the distributional gap between the selected set and the unselected pool (Chen et al., 2010).

D ADDITIONAL METHOD ILLUSTRATIONS

We provide enlarged illustrations of the two components described in Sections 4.1

D.1 CASE-BASED MEMORY-AUGMENTED ROUTING

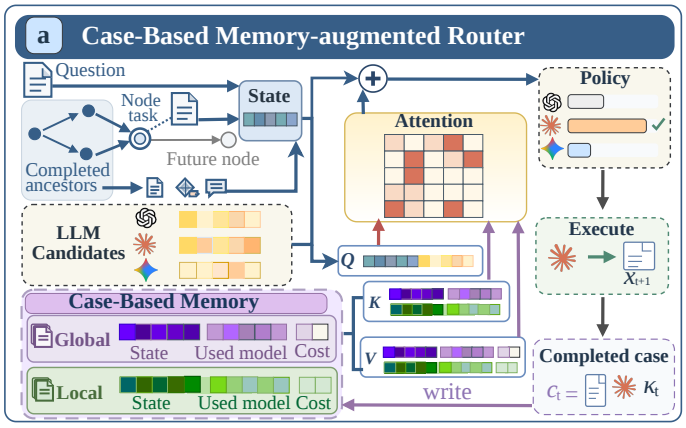


Figure 4: Case-based memory-augmented routing. The current execution state and each candidate LLM representation jointly query local and global memory. Retrieved case summaries augment the state for model selection. After execution, the completed case records the state, selected model, and execution cost. Global memory is updated during training and frozen during inference, while local memory is populated within each execution trace.

D.2 CANDIDATE LLM SET CONSTRUCTION

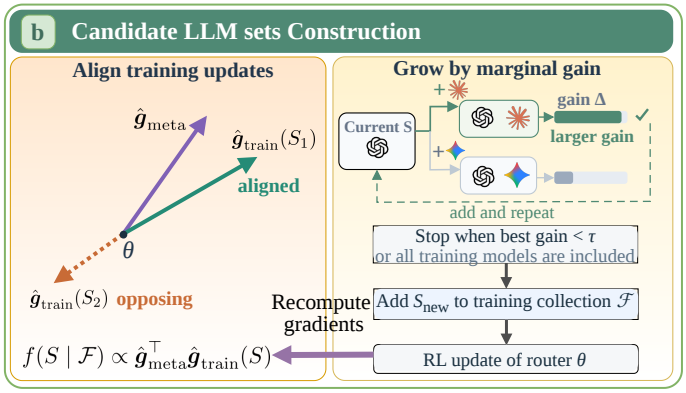


Figure 5: Candidate LLM set construction. The selection score measures alignment between the training gradient on a candidate set and the meta-gradient on the validation LLM pool. Each set is grown greedily by adding the model with the largest marginal gain until the gain falls below τ or all training models are included. The constructed set is added to the training collection, after which the router is retrained and gradients are recomputed.

E Dataset and LLM Backbone Details

Tables 5 and 6 summarize the benchmarks and executor pools used in our experiments. Following the multi-domain evaluation design of GraphPlanner (Feng et al., 2026), our suite covers five categories and 11 benchmarks. Within each category, source benchmarks provide training, validation, and in-domain test questions, while a separate benchmark is reserved for cross-dataset evaluation. Training, development, and unseen-evaluation executor pools are separated by model identity.

Table 5: **Task categories and benchmark roles.** Source benchmarks provide training, validation, and in-domain test questions; held-out benchmarks are used only for cross-dataset evaluation.

Category	Source benchmarks	Held-out benchmark
Knowledge	MMLU-Pro	GPQA-Diamond
Math	GSM8K, MATH	AIME 2024
Code	MBPP	HumanEval
Multi-hop QA	HotpotQA-Distractor	MuSiQue-Ans
Commonsense & Logic	CommonsenseQA	LogicGrid

Table 6: **LLM executor pools.** Development models guide candidate-set and checkpoint selection. Unseen-evaluation models are excluded from both training and development executor pools.

Role	LLMs
Training	Qwen3-0.6B, Qwen3-1.7B, Qwen3-4B, Qwen3-14B; Qwen3.5-2B, Qwen3.5-4B, Qwen3.5-9B, Qwen3.5-27B
Development	Qwen3.5-0.8B, Qwen3-8B, Qwen3-32B
Unseen evaluation	GPT-4.1 Nano, GPT-4.1 Mini, GPT-4.1; GPT-5 Nano, GPT-5 Mini, GPT-5; Mistral Small 2503, Mistral Medium 2505, Mistral Large 3

E.1 Task Descriptions

Table 7 reports the sample counts used in our experiments, rather than the full sizes of the original benchmarks. We use 1,974 training questions, 250 validation questions, and 530 test questions, comprising 300 in-domain and 230 cross-dataset questions. All methods share the same fixed query partitions. Table 8 summarizes the primary evaluation metric for each benchmark.

Table 7: **Sample counts by benchmark.** Underlined benchmarks are reserved for cross-dataset evaluation. Counts refer to original questions.

Category	Benchmark	Train	Validation	Test
Knowledge	MMLU-Pro	400	50	50
	GPQA-Diamond	0	0	50
Math	GSM8K	200	25	50
	MATH	400	50	50
	AIME 2024	0	0	30
Code	MBPP	374	50	50
	HumanEval	0	0	50
Multi-hop QA	HotpotQA-Distractor	400	50	50
	MuSiQue-Ans	0	0	50
Commonsense & Logic	CommonsenseQA	200	25	50
	LogicGrid	0	0	50
Total		**1,974**	**250**	**530**

Table 8: **Primary evaluation metrics.** The same benchmark-specific metric is used for all compared routers.

Category	Benchmark	Metric
Knowledge	MMLU-Pro	Accuracy
	GPQA-Diamond	Accuracy
Math	GSM8K	Numeric accuracy
	MATH	Mathematical equivalence accuracy
	AIME 2024	Numeric accuracy
Code	MBPP	Pass@1
	HumanEval	Pass@1
Multi-hop QA	HotpotQA-Distractor	Answer F1
	MuSiQue-Ans	Answer F1
Commonsense & Logic	CommonsenseQA	Accuracy
	LogicGrid	Accuracy

MMLU-Pro. MMLU-Pro evaluates academic knowledge and reasoning through multiple-choice questions across disciplines (Wang et al., 2024). Compared with MMLU, it emphasizes more demanding reasoning and expands the answer-choice set to up to ten options. We use it as the source Knowledge benchmark and report answer accuracy.

GPQA-Diamond. GPQA contains expert-authored multiple-choice questions in biology, chemistry, and physics, targeting specialized scientific understanding and reasoning (Rein et al., 2024). We use the Diamond subset as the held-out Knowledge benchmark to assess transfer from MMLU-Pro to expert-level science questions, evaluated using answer accuracy.

GSM8K. GSM8K comprises linguistically diverse grade-school mathematical word problems requiring multi-step arithmetic reasoning (Cobbe et al., 2021). Solutions combine relevant quantities and intermediate calculations to obtain a final numeric answer. We use it as a source Math benchmark and evaluate final numeric-answer correctness.

MATH. MATH contains competition-level problems spanning algebra, geometry, number theory, and other topics, with worked solutions in the original dataset (Hendrycks et al., 2021). It complements GSM8K with symbolic manipulation and more advanced problem solving. We use it as a source Math benchmark and score answers using mathematical equivalence checking.

AIME 2024. The American Invitational Mathematics Examination (AIME) poses competition mathematics problems with integer answers from 0 to 999 (Mathematical Association of America, n.d.). Unlike multiple-choice questions, these problems require deriving the answer without selecting from alternatives. We include all 30 problems from AIME 2024 as the held-out Math benchmark and report numeric accuracy.

MBPP. MBPP evaluates short Python program synthesis from natural-language specifications, using tasks accessible to entry-level programmers (Austin et al., 2021). Execution-based tests assess functional correctness rather than textual similarity to reference programs. We use it as the source Code benchmark and report pass@1.

HumanEval. HumanEval measures Python function completion from function signatures and docstrings (Chen et al., 2021). Generated solutions are checked against hidden tests to assess whether they implement the required behavior. It serves as the held-out Code benchmark for transfer from MBPP, with pass@1 as the metric.

HotpotQA-Distractor. HotpotQA requires integrating evidence across supporting documents to answer a question (Yang et al., 2018). The Distractor setting includes irrelevant context alongside relevant evidence, testing evidence selection and multi-hop reasoning. We use it as the source Multi-hop QA benchmark and report answer-level F1.

MuSiQue-Ans. MuSiQue-Ans contains answerable multi-hop questions composed from connected single-hop questions, with each reasoning step depending on information from another (Trivedi et al., 2022). This construction limits shortcuts based on disconnected evidence. Our held-out subset covers two-, three-, and four-hop examples, evaluated using answer-level F1.

CommonsenseQA. CommonsenseQA is a multiple-choice benchmark built using semantic relations from ConceptNet (Talmor et al., 2019). Questions require everyday background knowledge to distinguish between plausible alternatives rather than retrieving an answer from a passage. We use it as the source Commonsense & Logic benchmark and report accuracy.

LogicGrid. LogicGrid evaluates deduction from textual clues constraining relationships among entities and attributes (Mitra & Baral, 2015). A solution must identify an assignment consistent with the clues, testing constraint tracking and logical consistency. It is the held-out Commonsense & Logic benchmark, evaluated using accuracy.

E.2 Model Descriptions

Our inventory includes eight training and three development Qwen models, plus nine GPT-4.1, GPT-5, and Mistral models reserved for unseen-model evaluation. Table 9 reports configured parameter scales and input/output token rates; the three executor pools are disjoint.

Table 9: **Model sizes and input/output token rates.** Rates are per million tokens. Qwen rates are fixed synthetic prices; GPT-4.1 and Mistral rates are the frozen USD rates used in the experiments. GPT-5 rates follow OpenAI's standard (non-Batch) API pricing in USD for uncached input and output (OpenAI, 2025b). A dash indicates a parameter size not specified in our API configuration.

Role	Model	Size	Input	Output
Training	Qwen3-0.6B	0.6B	0.05	0.05
	Qwen3-1.7B	1.7B	0.10	0.10
	Qwen3-4B	4B	0.20	0.20
	Qwen3-14B	14B	0.80	0.80
	Qwen3.5-2B	2B	0.15	0.15
	Qwen3.5-4B	4B	0.20	0.20
	Qwen3.5-9B	9B	0.80	0.80
	Qwen3.5-27B	27B	2.40	2.40
Development	Qwen3.5-0.8B	0.8B	0.075	0.075
	Qwen3-8B	8B	0.40	0.40
	Qwen3-32B	32B	1.20	1.20
Unseen evaluation	GPT-4.1 Nano	–	0.10	0.40
	GPT-4.1 Mini	–	0.40	1.60
	GPT-4.1	–	2.00	8.00
	GPT-5 Nano	–	0.05	0.40
	GPT-5 Mini	–	0.25	2.00
	GPT-5	–	1.25	10.00
	Mistral Small 2503	–	0.10	0.30
	Mistral Medium 2505	–	0.40	2.00
	Mistral Large 3	–	0.50	1.50

Qwen3. Qwen3 is an open-weight model family supporting language, reasoning, and coding tasks (Yang et al., 2025). We use dense variants from 0.6B to 32B parameters: 0.6B, 1.7B, 4B, and 14B for training, and 8B and 32B for development. Their differing capacities and configured costs provide heterogeneous routing choices.

Qwen3.5. Qwen3.5 extends the family with native vision-language modeling (Qwen Team, 2026). We use 2B, 4B, 9B, and 27B variants for training and 0.8B for development; all process text inputs in our experiments.

GPT-4.1. GPT-4.1 emphasizes instruction following, coding, and long-context processing (OpenAI, 2025a). GPT-4.1 Nano, GPT-4.1 Mini, and GPT-4.1 provide three API-based executor choices with different token prices. They are introduced only during final unseen-model evaluation.

GPT-5. The GPT-5 API family includes GPT-5, GPT-5 Mini, and GPT-5 Nano, with smaller variants providing lower-cost alternatives (OpenAI, 2025b). All three are reserved for unseen-model evaluation and complement GPT-4.1 in assessing transfer across model families and generations.

Mistral. We additionally evaluate Mistral Small 2503, Medium 2505, and Large 3 (Mistral AI, 2025a;b;c). Their configured execution identifiers are `mistral-small-2503`, `mistral-medium-2505`, and `Mistral-Large-3`, respectively. They extend unseen-model evaluation beyond the Qwen and OpenAI families.

F Implementation Details

GREAT augments existing routers with case-based memory and candidate-set construction, without prescribing a common optimizer. Following Section 3.2, the integration requires a differentiable routing loss whose gradients can be estimated using the backbone's own learning procedure. We retain backbone-specific routing objectives and update rules rather than replacing them with a single reinforcement-learning formulation.

F.1 Adaptation to Routing Baselines

Shared interface. For each backbone, we use its current state representation x_t and candidate representation e_i to query case-based memory, following Section 4.1.2. Retrieved summaries are projected into the backbone's state space to obtain $\tilde{x}_{t,i}$, which replaces x_t when scoring candidate i. The integration therefore operates at the model-selection interface, without requiring identical encoders, hidden dimensions, or execution structures across backbones.

GraphRouter. GraphRouter uses supervised query–LLM edge prediction for routing (Feng et al., 2025). In our enhanced implementation, we replace its historical-interaction memory component with case-based retrieval, while retaining the query–LLM scoring interface and edge-prediction objective.

ICL-Router. ICL-Router learns to predict whether a candidate model can answer a query correctly using a supervised cross-entropy objective (Wang et al., 2026). We replace the capability-context memory used for routing with retrieved case representations, projected into the router's input space, while retaining the correctness-prediction objective.

GraphPlanner. GraphPlanner performs step-level agentic routing with a reinforcement-learning objective (Feng et al., 2026). We replace its routing memory with our local and global case banks, while retaining the workflow-state interface and the backbone's policy-optimization procedure.

RouterDC. RouterDC learns query–model compatibility through dual contrastive learning (Chen et al., 2024b). Since it has no standalone memory component in our implementation, we add case-based retrieval before query–model scoring and train the added module through the backbone's contrastive objective.

Router-R1. Router-R1 interleaves reasoning, model invocation, and response aggregation through reinforcement learning (Zhang et al., 2025). We add case-based memory at its model-selection steps, conditioning routing on the current reasoning state and retrieved cases. Its multi-round execution procedure and native learning objective are retained.

F.2 Candidate-Set Construction with Backbone-Specific Training

All five enhanced variants use the candidate-set construction procedure in Algorithm 2. For each backbone, $\mathcal{L}$ denotes its own routing loss: training gradients are estimated on candidate sets drawn from $\mathcal{M}_{\text{train}}$, and meta-gradients on the disjoint pool $\mathcal{M}_{\text{val}}$. The resulting alignment score guides greedy set construction. We alternate between adding a constructed set to the training collection and updating the augmented router over that collection. Trainable memory parameters receive the backbone's routing learning signal jointly with the router parameters. Learning rates, batch sizes, and update schedules remain backbone-specific.

Development pools guide candidate-set and checkpoint selection, while final unseen-model evaluation is kept separate. Each backbone and its enhanced counterpart use the same query partitions, executor endpoints, prompts, and scoring backends for their paired comparison. Parameters and memory banks are maintained separately for each method and cost setting.

F.3 Memory Management and Execution

Memory entries record completed decisions (x_j, e_j, κ_j) and are indexed by the candidate set under which they were collected. Global memory holds up to 512 cases, with training records committed after each batch; it remains read-only during validation and testing. Local memory is reset for each question and populated as execution proceeds. Single-round routers use global cases for their query-level choice, whereas multi-round routers additionally retrieve completed decisions from the current trace.

The implementation uses PyTorch, with router-side computation on a single NVIDIA RTX PRO 6000 Blackwell Server Edition GPU and executor LLMs accessed through separately served endpoints. Batched computation, embedding caches, and exact execution-response caches reduce repeated work. Cache hits retain their logical token costs, so caching does not alter the reported execution-cost metric.

G Additional Experimental Results

G.1 Routing performance without unseen LLMs

We complement the unseen-LLM evaluation in Section 5 with results obtained using identical training and test LLM pools, i.e., $\mathcal{M}_{\text{train}} = \mathcal{M}_{\text{test}}$. As shown in Table 10, GREAT improves average accuracy by 0.6–6.9 percentage points while reducing average execution cost by 14.8%–35.5% across all five routing backbones. These improvements hold for both single-round and multi-round routers, showing that the benefits of GREAT also extend to the seen-LLM setting.

Table 10: Routing performance comparison with router baselines under seen LLMs ($\mathcal{M}_{\text{train}} = \mathcal{M}_{\text{test}}$). **Bold** indicates the better value within each backbone pair (higher Acc and lower Cost); for ties, only the GREAT-enhanced variant is highlighted. Δ Avg. reports the change introduced by GREAT, where Δ Acc is measured in percentage points (pp) and Δ Cost denotes the relative change in average cost (↓ is better).

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average		Δ Avg.	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Δ Acc (pp)	Δ Cost (%)
Single-round Router														
RouterDC	72.0%	62.0	69.0%	41.9	74.0%	82.4	79.0%	36.7	66.0%	24.5	72.0%	49.5	–	–
RouterDC+GREAT	**72.0%**	**62.0**	**74.0%**	**35.5**	**74.0%**	**56.8**	**79.0%**	**36.7**	**82.0%**	**19.9**	**76.2%**	**42.2**	**+4.2**	**-14.8%**
GraphRouter	62.0%	54.6	51.0%	14.0	40.0%	36.2	42.8%	17.6	40.0%	**5.5**	47.2%	25.6	–	–
GraphRouter+GREAT	**66.0%**	**52.2**	**61.0%**	**12.4**	**44.0%**	**18.9**	**51.4%**	**11.3**	**48.0%**	5.6	**54.1%**	**20.1**	**+6.9**	**-21.4%**
ICL-Router	88.0%	28.5	82.0%	37.6	58.0%	22.4	75.0%	28.6	80.0%	12.5	76.6%	25.9	–	–
ICL-Router+GREAT	**88.0%**	**11.4**	**92.0%**	**25.9**	**62.0%**	**21.0**	**78.0%**	**25.9**	**82.0%**	**11.6**	**80.4%**	**19.2**	**+3.8**	**-26.1%**
Multi-round Router														
Router-R1	23.2%	102.9	17.5%	28.2	30.0%	89.7	13.7%	42.0	26.0%	28.9	22.1%	58.3	–	–
Router-R1+GREAT	**26.5%**	**51.4**	**18.0%**	**18.7**	**33.0%**	**84.7**	**17.5%**	**22.3**	**26.0%**	**10.9**	**24.2%**	**37.6**	**+2.1**	**-35.5%**
GraphPlanner	76.0%	27.6	92.0%	45.8	76.0%	128.7	86.3%	123.5	84.0%	17.4	82.9%	68.6	–	–
GraphPlanner+GREAT	**78.0%**	**25.1**	**93.0%**	**34.2**	**76.0%**	**63.4**	**86.3%**	**123.5**	**84.0%**	**15.7**	**83.5%**	**52.4**	**+0.6**	**-23.6%**

Generalization to unseen datasets. Table 11 evaluates the same LLM pool on the five held-out datasets described in Section 5.1. GREAT reduces average execution cost by 4.3%–48.2% across all five backbones and improves average accuracy by 3.6–8.2 percentage points for four of them. Router-R1 is the exception: its average accuracy decreases by 2.3 percentage points while its average execution cost falls by 48.2%.

Table 11: Performance on unseen datasets (GPQA, AIME, HumanEval, MuSiQue-Ans, and Logic-Grid) for evaluating zero-shot generalization with seen LLMs. **Bold** indicates the better value within each backbone pair (higher Acc and lower Cost); for ties, only the GREAT-enhanced variant is highlighted. Δ Avg. reports the change introduced by GREAT, where Δ Acc is measured in percentage points (pp) and Δ Cost denotes the relative change in average cost ($\downarrow$ is better).

Router	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average		Δ Avg.	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Δ Acc (pp)	Δ Cost (%)
RouterDC	50.0%	71.0	20.0%	47.5	86.0%	20.0	37.0%	54.9	64.0%	40.8	51.4%	46.8	–	–
RouterDC+GREAT	**50.0%**	**71.0**	**20.0%**	**47.5**	**88.0%**	**18.0**	**37.0%**	**54.9**	**80.0%**	**32.0**	**55.0%**	**44.7**	**+3.6**	**-4.6%**
GraphRouter	36.0%	25.6	10.0%	22.9	40.0%	20.0	16.8%	30.2	60.0%	50.5	32.6%	29.9	–	–
GraphRouter+GREAT	**44.0%**	**23.2**	**20.0%**	**21.0**	**48.0%**	**18.1**	**20.0%**	**21.0**	**72.0%**	**48.8**	**40.8%**	**26.4**	**+8.2**	**-11.6%**
ICL-Router	82.0%	23.0	53.3%	**18.9**	78.0%	11.5	59.8%	81.6	94.0%	28.1	73.4%	32.6	–	–
ICL-Router+GREAT	**84.0%**	**22.5**	**60.0%**	19.6	**90.0%**	**11.3**	**69.1%**	**76.7**	**96.0%**	**25.9**	**79.8%**	**31.2**	**+6.4**	**-4.3%**
Router-R1	**22.0%**	229.7	10.0%	72.0	42.0%	72.3	**7.5%**	38.7	**27.0%**	240.5	**21.7%**	130.6	–	–
Router-R1+GREAT	16.0%	**99.3**	**10.0%**	**72.0**	**42.0%**	**42.1**	5.1%	**15.2**	24.0%	**110.0**	19.4%	**67.7**	**-2.3**	**-48.2%**
GraphPlanner	80.0%	628.5	63.3%	220.2	84.0%	26.9	**69.1%**	744.1	100.0%	284.2	79.3%	380.8	–	–
GraphPlanner+GREAT	**84.0%**	**565.3**	**80.0%**	**191.5**	**94.0%**	**26.1**	67.9%	**682.9**	**100.0%**	**276.5**	**85.2%**	**348.4**	**+5.9**	**-8.5%**

G.2 Effect of case-based memory

Table 12 reports the numerical results underlying Figure 3(a). Removing local and global memory reduces average accuracy by 3.1 and 11.3 percentage points, respectively, and increases average execution cost from 68.4 to 76.4 and 112.4.

Table 12: Effect of local and global memory in GraphPlanner+GREAT. These results are visualized in Figure 3(a).

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
GraphPlanner+GREAT w/o local	83.7%	117.0	85.0%	36.5	**82.0%**	106.4	82.8%	88.4	80.0%	33.7	82.7%	76.4
GraphPlanner+GREAT w/o global	77.4%	148.0	79.0%	131.0	78.0%	141.4	66.4%	96.3	72.0%	45.3	74.5%	112.4
GraphPlanner+GREAT	**84.0%**	113.3	**92.0%**	51.6	**82.0%**	73.9	**89.1%**	75.5	**82.0%**	27.6	**85.8%**	68.4

G.3 Effect of candidate LLM set selection

Disabling candidate set selection reduces average accuracy by 10.8 percentage points, with nearly unchanged average execution cost (Table 2). Table 13 reports the numerical results underlying Figure 3(b). Increasing K from 2 to 4 and then 6 raises average accuracy from 79.3% to 85.3% and 86.2%, respectively, with diminishing gains; average execution cost varies non-monotonically.

Table 13: Effect of the number of training candidate sets (K in Algorithm 2) for GraphPlanner+GREAT. These results are visualized in Figure 3(b).

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
$K = 2$	79.6%	64.4	83.0%	35.9	72.0%	119.6	86.1%	28.4	76.0%	11.5	79.3%	52.0
$K = 4$	**84.0%**	117.2	90.0%	45.6	80.0%	124.2	88.6%	74.7	**84.0%**	42.9	85.3%	80.9
$K = 6$	**84.0%**	113.3	**92.0%**	51.6	**82.0%**	73.9	**89.1%**	75.5	**84.0%**	27.6	**86.2%**	68.4

G.4 Effect of LLM sets used in J_{OUT}

Table 14 reports the numerical results underlying Figure 3(c). Across the tested validation pools, average accuracy ranges from 82.2% to 83.0%, a spread of only 0.8 percentage points, while average execution cost ranges from 84.3 to 92.6. These results indicate limited sensitivity of average routing accuracy to validation-pool composition and size within the evaluated range.

Table 14: Effect of the validation LLM pool used in J_{out} for GraphPlanner+GREAT. These results are visualized in Figure 3(c).

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
$\lvert\mathcal{M}_{\text{val}}\rvert = 3$ (Set 1)	84.0%	113.3	89.0%	62.0	76.0%	166.3	85.1%	74.7	**78.0%**	46.9	82.4%	92.6
$\lvert\mathcal{M}_{\text{val}}\rvert = 3$ (Set 2)	85.4%	116.2	**91.7%**	50.9	76.6%	139.2	83.5%	73.4	**78.0%**	41.6	**83.0%**	84.3
$\lvert\mathcal{M}_{\text{val}}\rvert = 4$	**85.7%**	115.8	87.4%	59.7	76.6%	141.7	**85.7%**	75.9	**78.0%**	44.8	82.7%	87.6
$\lvert\mathcal{M}_{\text{val}}\rvert = 5$	83.7%	117.0	88.5%	46.5	**77.0%**	136.4	83.8%	88.4	**78.0%**	33.7	82.2%	84.4

G.5 Effect of the size of global memory

Figure 6 compares global memory budgets of 64, 128, and 256 cases for GraphPlanner+GREAT; Table 15 gives the per-category results. Average accuracy increases from 75.7% to 80.9% and 82.6%, respectively. The two successive doublings yield gains of 5.2 and 1.7 percentage points, showing diminishing accuracy gains as memory grows. Average execution cost varies non-monotonically, from 89.4 to 90.2 and then 77.5. Among the tested sizes, 256 cases therefore achieves both the highest average accuracy and the lowest average execution cost.

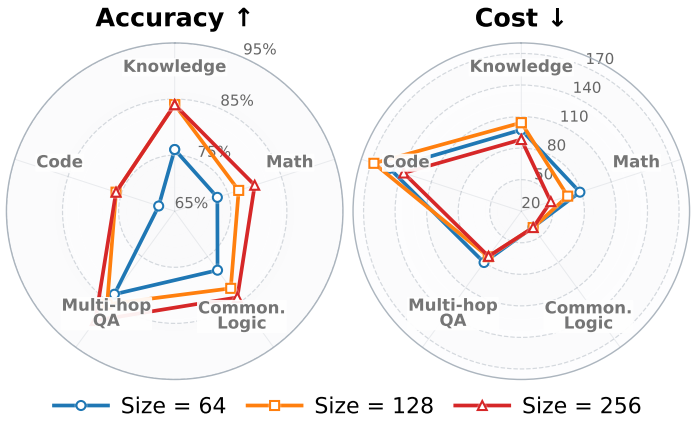


Figure 6: Effect of global memory size on GraphPlanner+GREAT. The plots show accuracy (left, higher is better) and execution cost (right, lower is better) across five task categories for memory budgets of 64, 128, and 256 cases.

Table 15: Routing performance with different global memory sizes.

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
Size = 64	76.0%	97.4	73.0%	78.6	68.0%	151.8	83.3%	80.3	78.0%	38.8	75.7%	89.4
Size = 128	**84.0%**	104.2	77.0%	66.4	**76.0%**	167.4	85.5%	73.5	82.0%	39.2	80.9%	90.2
Size = 256	**84.0%**	88.2	**80.0%**	49.5	**76.0%**	137.6	**89.1%**	73.0	**84.0%**	39.2	**82.6%**	77.5

G.6 Detailed candidate-set selection results

Table 16 provides the per-category results summarized in Table 2. All strategies use GraphPlanner with case-based memory, including *No*, which disables candidate-set selection.

Table 16: Routing performance comparison with varied ways of selecting candidate sets for training (taking GraphPlanner as the backbone and enabling case-based memory for all methods)

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
No	74.0%	78.0	75.0%	25.8	70.0%	90.4	86.0%	77.8	76.0%	26.1	76.2%	59.6
Random	76.0%	96.7	75.0%	32.5	68.0%	106.0	86.3%	67.7	78.0%	49.9	76.7%	70.5
Singleton	**84.0%**	119.0	89.0%	67.0	76.0%	167.4	**89.1%**	75.5	78.0%	49.9	83.2%	95.8
Sim-Coverage	82.0%	93.0	89.0%	67.0	72.0%	148.7	88.4%	74.7	78.0%	49.9	81.9%	86.6
Sim-diversity	**84.0%**	119.0	89.0%	67.0	76.0%	167.4	**89.1%**	75.5	82.0%	29.0	84.0%	91.6
DPP	**84.0%**	119.0	89.0%	67.0	76.0%	167.4	**89.1%**	75.5	78.0%	49.9	83.2%	95.8
MMD	82.0%	93.0	89.0%	67.0	74.0%	166.3	88.4%	55.3	86.0%	27.6	83.9%	81.9
GREAT	**84.0%**	83.3	**92.0%**	51.6	**82.0%**	73.9	**89.1%**	68.8	**88.0%**	19.6	**87.0%**	59.4

G.7 Generalization to unseen LLMs and datasets

Table 17 provides the per-category results summarized in Table 3. This setting uses a disjoint test LLM pool, $\mathcal{M}_{\text{train}} \cap \mathcal{M}_{\text{test}} = \emptyset$, and the five held-out datasets: GPQA-Diamond, AIME 2024, HumanEval, MuSiQue-Ans, and LogicGrid.

Table 17: Performance on unseen datasets (GPQA, AIME, HumanEval, MuSiQue-Ans, and LogicGrid) for evaluating zero-shot generalization with unseen LLMs. **Bold** indicates the better value within each backbone pair (higher Acc and lower Cost); for ties, only the GREAT-enhanced variant is highlighted. Δ Avg. reports the change introduced by GREAT, where Δ Acc is measured in percentage points (pp) and Δ Cost denotes the relative change in average cost ($\downarrow$ is better).

Router	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average		Δ Avg.	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Δ Acc (pp)	Δ Cost (%)
RouterDC	58.0%	27.0	40.0%	51.4	98.0%	25.2	65.2%	51.7	54.0%	36.3	63.0%	38.3	–	–
RouterDC+GREAT	**62.0%**	**24.0**	**50.0%**	**15.7**	**98.0%**	**15.7**	**67.0%**	**50.7**	**58.0%**	**26.4**	**67.0%**	**26.5**	**+4.0**	**-30.9%**
GraphRouter	62.0%	61.5	36.7%	87.8	88.0%	48.7	60.2%	58.0	75.0%	14.5	64.4%	54.1	–	–
GraphRouter+GREAT	**68.0%**	**56.3**	**43.3%**	**78.8**	**92.0%**	**14.0**	**61.1%**	**39.9**	**78.0%**	**14.1**	**68.5%**	**40.6**	**+4.1**	**-24.9%**
ICL-Router	58.0%	17.7	28.0%	50.4	96.0%	52.4	54.6%	74.5	84.0%	67.0	64.1%	52.4	–	–
ICL-Router+GREAT	**60.0%**	**14.4**	**37.0%**	**10.9**	**96.0%**	**52.4**	**58.7%**	**13.0**	**90.0%**	**20.3**	**68.3%**	**22.2**	**+4.2**	**-57.6%**
Router-R1	24.0%	22.0	7.0%	46.1	44.0%	14.6	14.0%	141.5	38.0%	121.9	25.4%	69.2	–	–
Router-R1+GREAT	**34.0%**	**13.8**	**13.0%**	**42.6**	**50.0%**	**14.2**	**14.2%**	**89.7**	**42.0%**	**117.3**	**30.6%**	**55.5**	**+5.2**	**-19.8%**
GraphPlanner	70.0%	135.9	43.3%	**175.2**	92.0%	62.1	58.4%	160.0	80.0%	167.8	68.8%	140.2	–	–
GraphPlanner+GREAT	**74.0%**	**129.2**	**46.7%**	176.2	**96.0%**	**53.4**	**61.7%**	**123.9**	**82.0%**	**161.0**	**72.1%**	**128.7**	**+3.3**	**-8.2%**

G.8 Transductive vs. inductive

Table 18 provides the per-category results summarized in Table 4. The inductive setting does not reuse global memory collected during training, whereas the transductive setting retains it for inference. For GraphPlanner+GREAT, reusing this memory improves average accuracy from 82.8% to 83.2% and reduces average execution cost from 97.4 to 91.5.

Table 18: Inductive and transductive routing inference. The two settings differ in whether global memory collected during training is reused at inference. Bold highlights the accuracy of the final routing configuration.

Setting	Knowledge		Math		Code		Multi-hop QA		Common. & Logic		Average	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
GraphPlanner-ind	74.0%	38.2	75.0%	21.8	70.0%	54.6	86.0%	29.8	82.0%	15.1	77.4%	31.9
GraphPlanner-trans	74.0%	38.0	75.0%	19.4	70.0%	50.4	86.0%	27.8	82.0%	15.1	77.4%	30.1
GraphPlanner+GREAT-ind	84.0%	132.5	89.0%	67.0	76.0%	166.1	87.2%	74.6	78.0%	46.9	82.8%	97.4
GraphPlanner+GREAT-trans	**84.0%**	113.3	**89.0%**	67.0	**76.0%**	156.3	**89.1%**	74.7	78.0%	45.9	**83.2%**	91.5

G.9 Training efficiency analysis

Figure 7 compares validation reward over training steps for GraphPlanner, its memory-only variant, and GraphPlanner+GREAT. The full method reaches comparable reward levels in fewer steps and attains a higher final reward than both controls. Its validation reward stabilizes near 0.78 in the later training stages, while the memory-only variant and GraphPlanner finish at lower levels. The

improvement over the memory-only variant supports the additional benefit of candidate-set selection beyond case-based memory. These curves measure training-step efficiency for this backbone, rather than wall-clock training time.

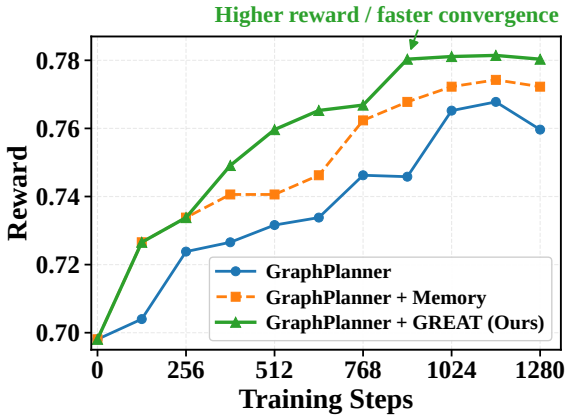


Figure 7: Validation reward during training. GraphPlanner+Memory denotes the memory-only variant; GraphPlanner+GREAT additionally uses candidate-set selection.

G.10 Accuracy–cost trade-off

Figure 8 compares GraphPlanner and GraphPlanner+GREAT on MMLU-Pro for execution-cost weights $\lambda \in \{0, 0.2, 0.5, 0.8, 1.0\}$. As λ increases, both routers reduce execution cost at the expense of accuracy. For every tested λ, GraphPlanner+GREAT achieves higher accuracy at lower execution cost than GraphPlanner. The improvements thus persist across both accuracy-oriented and cost-oriented settings, rather than depending on a single choice of λ.

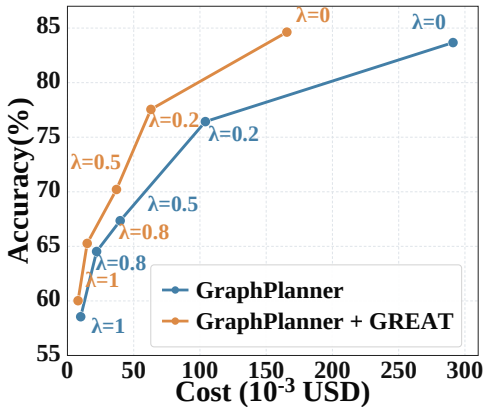


Figure 8: Accuracy–cost trade-offs on MMLU-Pro. Each point is labeled with its execution-cost weight λ; execution cost is shown in 10^{-3} USD.

G.11 Empirical assumption verification

We examine the first-order optimization residual, routing-probability gaps, and score-gradient norms of GraphPlanner+GREAT on MMLU-Pro.

G.11.1 Empirically Verifying Assumptions for Approximate Submodularity of $f(S)$

Verifying $\delta_{\min}$. We examine the empirical distribution of the top-1/top-2 log-probability gap,

$$\delta(x, S) = \log \pi_{\boldsymbol{\theta}}(m^*(x, S) \mid x, S) - \log \pi_{\boldsymbol{\theta}}(m^{(2)}(x, S) \mid x, S).$$

For the evaluated checkpoint and candidate-set configuration, the 5th percentile and median gaps are 2.344 and 4.869, respectively. These correspond to top-1/top-2 probability ratios of approximately 10.42 and 130.19. The substantial gap even at the 5th percentile indicates a clear preference for the top-ranked model across most observed routing decisions. These results provide empirical support for the confident-routing premise in the evaluated setting.

Verifying bounded reward assumption. Under maximum-cost normalization, raw execution costs are divided by a fixed maximum cost $C_{\max} > 0$ covering the available models under the prescribed execution budget. For example, for a pool containing Qwen3.5-2B and Qwen3.5-27B, the same normalization constant covers both models' execution costs under the allowed input and output token budgets. The normalized cost therefore satisfies

$$\mathrm{Cost}(m, x) = \frac{C(m, x)}{C_{\max}} \in [0, 1].$$

Since task utility also lies in $[0, 1]$, the reward satisfies

$$-\lambda \leq b_m(x) = U(m, x) - \lambda\, \mathrm{Cost}(m, x) \leq 1.$$

Thus, $|b_m(x)| \leq \max\{1, \lambda\}$, establishing bounded rewards under this normalization.

Verifying the bounded-gradient assumption. After 1,536 training trajectories, we evaluate

$$\|\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(m \mid x, S)\|_2$$

over 1,000 state–set–action combinations constructed from the root states of 128 report-validation questions, including non-top-1 actions. The median, 95th percentile, and observed maximum of the gradient norms are 19.45, 45.04, and 55.35, respectively. All evaluated gradients are finite, with no NaN or infinite values observed. These results provide an empirical gradient envelope over the inspected configurations and support the plausibility of the bounded-gradient assumption in the evaluated setting.