ROUTE-THEN-PROBE: A TWO-STAGE CASCADE FOR COST-EFFECTIVE LLM ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Routing algorithms reduce the cost of LLM service by selecting a suitable model for each request from a pool of models with varying sizes, capabilities, and prices. We observe that a well-matched small model can answer many requests at a fraction of the cost of a flagship model. However, even a well-matched small model can fail, so realizing these savings requires two decisions: which small model to select and whether to escalate to a stronger model. We propose Route-then-Probe (RTP), a two-stage cascade that first selects exclusively among small models and then uses the inference-time signals of the selected model to decide whether to escalate. In a single forward pass, a lightweight router jointly encodes each request and the features of all candidate small models to select the model most likely to answer correctly. A calibrated probe gate then examines the hidden states of the selected model and determines, before decoding, whether the estimated accuracy gain from escalation justifies its cost. Existing routers typically select directly from a mixed pool of small and flagship models using request-side evidence alone, which may be insufficient to predict per-request success. Hidden-state probes provide model-specific correctness signals, but probing every candidate incurs substantial inference overhead and requires access to the internals of each model. RTP exploits complementarity between small models while probing only the selected model, avoiding the overhead of probing the entire pool. Experiments with seven self-deployed small models and an API-based flagship model on a mixed dataset spanning more than ten domains show that RTP achieves better cost–accuracy trade-offs than existing routing and cascading methods across the evaluated cost range. Its modular architecture also accommodates existing routers as first-stage selectors, extending the benefits of selective probing to other routing algorithms.

1 INTRODUCTION

Frontier large language models (LLMs) keep growing in scale and capability. So does the cost of serving them: a request answered by a flagship model can be more than an order of magnitude more expensive than the same request answered by an open-weight model of 7–9B parameters (Achiam et al., 2023; Chen et al., 2024). Yet not every request needs the flagship. Small open-weight models such as Qwen (Yang et al., 2024), Llama (Dubey et al., 2024), Mistral (Jiang et al., 2023), and Gemma (Gemma Team et al., 2024) already answer a large share of requests correctly. They are cheap and fast to serve, and they are easier to fine-tune on proprietary, domain-specific data to become specialists, for example with LoRA (Hu et al., 2022). A serving system therefore typically maintains not a single model but a pool of models that differ in size, capability, price, and deployment mode, ranging from self-deployed open-weight models to paid API flagships (Hu et al., 2024). The quality and the cost of such a system then hinge on one decision for each incoming request: which model should answer it? Making this decision well is the task of an intelligent routing algorithm. It dispatches each request to a model that can handle it, trading accuracy against expense under a preference that may vary across users and applications.

Existing routing methods differ in where they look for their signal. Many of them predict each candidate's success from the request text and the models' outcomes on past requests (Moslem & Kelleher, 2026). kNN-Router (Shnitzer et al., 2023) retrieves the outcomes of semantically similar past requests, Avengers-Pro (Zhang et al., 2025c) aggregates them into cluster-level accuracy and price statistics, and IRT-Router (Song et al., 2025) fits model ability and query difficulty parameters

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.

from past answers. Such signals are cheap to obtain, since no LLM ever runs. The evidence records how the models behaved on historical requests but offers no view of a candidate processing the request at hand, so every prediction must generalize from that history to the current request. With limited supervision, per-request predictions of success remain error-prone (Varshney et al., 2026). In a mixed pool of small and flagship models, these errors are costly. Routing a hard request to a small model that cannot answer it commits an incorrect answer with no remedy, and sending a request to the flagship although a small model would have answered it correctly pays the flagship price unnecessarily.

Signals from a model's own inference are more specific to that model (Varshney et al., 2026). A probe is a lightweight classifier trained to read, from a model's internal states, information about whether it will answer the current request correctly. The signal can be taken from different points of inference (Zhang et al., 2025a). Linear probes on prefill hidden states decode such information before any answer token is generated (Moreno Cencerrado et al., 2026), and on the MATH benchmark they predict the success of a math-tuned small model better than TF–IDF and question-length features (Lugoloobi et al., 2026). Whatever the point of inference, obtaining such a signal requires a model to actually run on the request. This adds substantial compute and latency at routing time, and API-based models may not expose the required states at all. Probing every candidate, as the pool-routing policy of PIKA does, requires prefilling the whole pool before routing (Lugoloobi et al., 2026). This multiplies the prefill cost and the routing latency on every request, and an API flagship still cannot be probed at all. PIKA's cascade variant avoids the full-pool cost by fixing a single small model and probing only it: the probe judges whether this one model can answer the request, and requests it cannot answer are escalated to a stronger model. The other small models are never consulted. PrefillRouter (Varshney et al., 2026) avoids prefilling every candidate by predicting all of them from the states of a common encoder, a small open-weight model. The encoder's prefill hidden states yield a richer request feature, but that feature reflects the encoder reading the request, not any candidate processing it. Output-side cascades (Dekoninck et al., 2025) defer the decision until a model has generated an answer, so the generation is paid for before the decision. A router that uses such signals must therefore decide which models to run before routing and where in inference to read the signal, balancing their added cost and latency against the accuracy they bring.

1.1 Our Main Ideas and Contributions

Request-side signals are cheap but error-prone, while inference-side signals are accurate but expensive. How to use the two signals together efficiently has received little attention. We propose **Route-then-Probe** (RTP), a two-stage cascaded routing algorithm. A lightweight router (L1) selects, from request-side signals, the small model whose strengths match the request. A calibrated probe gate (L2) then uses the inference-side signal of that model alone, reading its prefill hidden states to decide whether the flagship must take over. Probing one well-chosen model instead of the whole pool is what makes the stronger signal affordable.

Figure 1 previews why this design pays off. Against the mixed-pool routers, our lead widens as the budget grows, because the probe signal decides more accurately which requests need the flagship. Against the fixed-model probe cascades, our strength is the request-side signal: it exploits the complementarity of the small models at zero escalation cost, and our router alone already matches the accuracy that a fixed model's own probe reaches only after escalating 10–27% of its requests to the flagship (stars). The full-pool policy of PIKA does not appear in the figure: it requires a probe for the flagship, which an API model does not expose, whereas our L2 needs no flagship probe at all.

Our contributions can be summarized as follows:

- We propose **Route-then-Probe** (RTP), a two-stage cascaded routing algorithm that serves each request cost-effectively: it selects among small models with request-side semantic signals and decides escalation with the inference-time signal of the selected model alone. On seven self-deployed small models and one API-based flagship over a mixed dataset spanning more than ten domains, RTP improves the average performance gap recovered (APGR), a cost-aware measure of how much of the accuracy gap between a weak and a strong model that a method closes across cost levels (Ong et al., 2025), by 4.4 points over the best baseline, and it is the best method across the evaluated cost range.

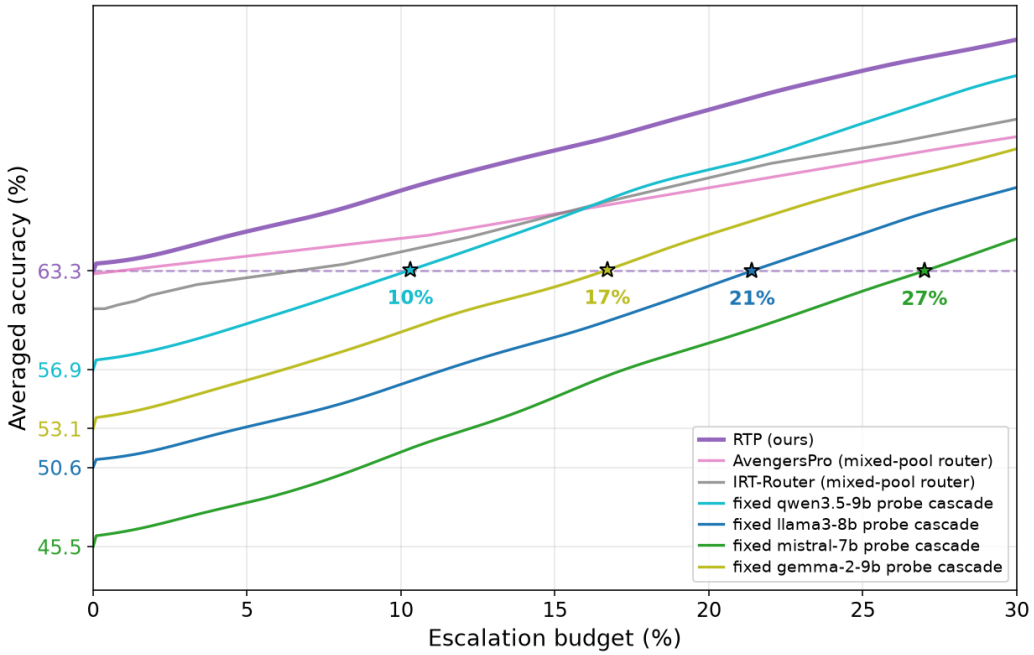


Figure 1: Accuracy versus the share of requests sent to the flagship. The pool contains four small models and one API-based flagship, evaluated on a mixed benchmark. RTP selects among the small models with L1 and escalates with L2. Mixed-pool routers run their native policies over the full pool. Each fixed-model probe cascade uses one fixed small model, whose probe decides escalation. Stars mark where the cascades first match the zero-escalation accuracy of L1.

- We design a lightweight L1 that jointly encodes each request and the features of all candidate small models with a single 149M-parameter encoder. Its training combines per-model correctness supervision with a pool-level listwise objective (Cao et al., 2007). Using request-side signals, L1 selects the most suitable small model with almost no inference overhead. In our component analysis, L1 attains the highest APGR among all deployable routers and comes very close to the domain oracle, a non-deployable upper bound that selects the best model from ground-truth domain labels.
- Our architecture can incorporate existing routers by having them select only among the small models while L2 decides escalation. The calibrated probe gate weighs the estimated accuracy gain against the cost. Under this architecture, every pool router in Table 1 improves over its native policy.

2 Related Work

We briefly review the most relevant methods and refer the reader to a recent survey (Moslem & Kelleher, 2026) for more details on LLM routing and cascading.

2.1 Routing among LLM pools

Pool routing methods learn a mapping from each incoming request to a model in the pool, differing mainly in how they summarize the models' past performance. TagRouter (Chen et al., 2025) converts the request into capability tags and picks the candidate with the best historical performance on those tags. EmbedLLM (Zhuang et al., 2025) learns compact model embeddings from a correctness matrix. Capability instruction tuning (Zhang et al., 2025b) turns a model's aptitude-test scores into a text profile, and a lightweight LLM reads this profile to judge whether the model can handle the request. ICL-Router (Wang et al., 2026) encodes each model as an in-context vector built from its

query–performance pairs on a small set of queries. HyDRA (Garg et al., 2026) predicts what a request needs along four hand-defined capability dimensions and matches it against a catalog of model ability vectors. Hybrid LLM (Ding et al., 2024) and RouteLLM (Ong et al., 2025) instead train a predictor over the request alone for a pair of one small and one large model, the former regressing the quality gap between the pair and the latter learning preference-based scores. CARROT (Somerstep et al., 2025) predicts both quality and cost for each candidate and optimizes their weighted combination, and MixLLM (Wang et al., 2025) routes a stream of requests under cost and latency constraints, updating its predictors online. ZeroRouter (Yan et al., 2026) onboards a new model into a universal latent space from a small set of anchor questions, without retraining the router. PrefillRouter (Varshney et al., 2026) is closest to using inference-time evidence, computing prefill hidden states with a separate open-weight encoder that acts as a feature extractor and predicting the correctness of every candidate from them.

2.2 Inference-time signals and escalation

A different class of methods reads inference-time signals, which become available only once the model starts computing. The prefill pass encodes the request before the first token is emitted, and the decode pass keeps producing hidden states and text as the answer is written. Language models carry internal evidence about their own correctness: they can assess what they know (Kadavath et al., 2022), and their hidden states encode whether a statement they generate is true or false (Azaria & Mitchell, 2023). Probes turn this into a routing signal. Linear probes over the question's prefill hidden states predict, without generating an answer, whether the model will answer the question correctly (Moreno Cencerrado et al., 2026). PIKA (Lugoloobi et al., 2026) trains such probes for each model in a pool and routes queries according to the predicted success; its cascade variant instead probes one small model and escalates to a stronger one at a threshold. Probes also work during generation: chunk-level probes on intermediate reasoning steps predict whether the final answer will be correct and support early exit (Zhang et al., 2025a). These methods require access to the probed models' hidden states, at the prefill or at intermediate decoding steps. Output-side cascades instead judge the generated answer. FrugalGPT (Chen et al., 2024) learns a cascade of API models and escalates when a trained scorer is not confident in the answer. AutoMix (Aggarwal et al., 2024) lets a small model self-verify its draft answer before escalating. CascadeRouting (Dekoninck et al., 2025) unifies routing and cascading into a single threshold rule on predicted quality and cost, re-deciding among the remaining models whenever an answer is judged insufficient.

3 Our Proposed Method

In this section, we propose Route-then-Probe (RTP), a two-stage cascaded routing algorithm that routes each request to the small model best suited to answer it and escalates to the flagship only when the selected model is likely to fail. Figure 2 gives an overview of the architecture. We study a serving system with a pool of M self-deployed small models, denoted by $m_1, \ldots, m_M$, and one API-based flagship F. We assume a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ of N requests, where x_i is the full prompt and $y_i \in \{0,1\}^M$ records the results of the small models, with $y_{i,m} = 1$ if model m answers x_i correctly. For any request x, L1 reads it and selects the small model $m^*(x)$ best suited to answer it. L2 then reads the prefill hidden states that $m^*(x)$ computes as part of serving, estimates whether it will answer x correctly, and escalates to the flagship only when the estimated accuracy gain justifies the cost.

3.1 L1: A Lightweight Pool Router

Our design follows a natural idea: a learned router should see the request and every model's features together, both when it is trained and when it routes. Its decision can then depend on the interaction between what the request asks and what each model is good at.

Concretely, we sample K requests uniformly at random from the training set, with $K \ll N$, and record whether each model answers them correctly. The fingerprint of model m is this correctness record, $\varphi_m \in \{0,1\}^K$, where $\varphi_{m,k} = 1$ if model m answers the k-th sampled request correctly. The router places every fingerprint next to the request in one sequence,

$$a_m = b_m + W_\varphi \, \varphi_m, \qquad \text{input} = [\, e_{\text{CLS}},\ a_1, \ldots, a_M,\ \text{emb}(x)\,], \tag{1}$$

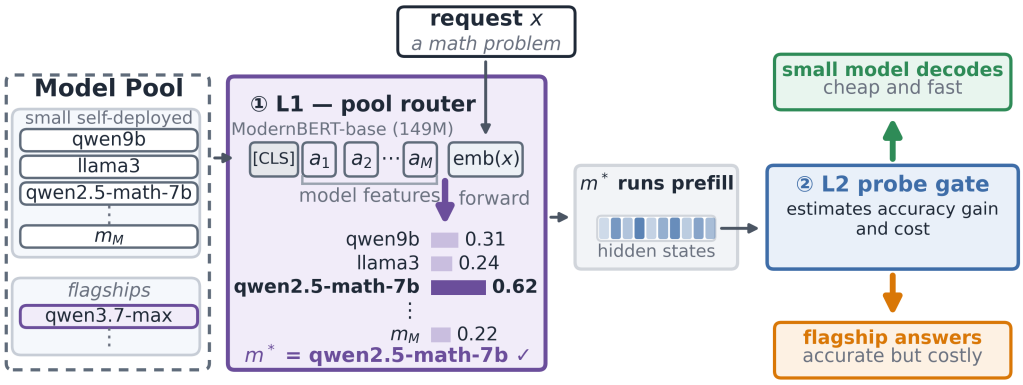


Figure 2: The Route-then-Probe architecture. L1 selects the best-suited small model $m^*(x)$ in one forward pass, and L2 probes the prefill hidden states of m^* to decide whether to escalate to the flagship.

where $b_m \in \mathbb{R}^d$ is a learned bias for model m and $W_\varphi \in \mathbb{R}^{d\times K}$ projects the fingerprint into the encoder's hidden space, both trained jointly with the encoder. Here $\mathrm{emb}(x)$ denotes the token embeddings of the request. A shared encoder, ModernBERT-base (Warner et al., 2025) of 149M parameters, encodes the sequence. Through self-attention, every model token and every request token exchange information, so the two sides interact within a single forward pass.

Mapping each incoming request to the most suitable model in the pool parallels search systems, which map a query to the most relevant items in a corpus (Wang et al., 2011). Retrieval encodes the query and each item independently with a dual-encoder (Karpukhin et al., 2020) and reduces each comparison to a single similarity computation (Humeau et al., 2020). It is fast enough to screen millions of items, but coarse. Re-ranking re-scores a shortlist with a cross-encoder that jointly models the query and the item (Nogueira et al., 2020), so the score reflects their fine-grained interaction, at a cost that only a shortlist can afford. Many existing routers sit on the retrieval side of this spectrum. IRT-Router scores a candidate by the dot product of a request vector and a model ability vector. EmbedLLM scores it by a linear head over the elementwise product of the request embedding and the model embedding. Because a model pool keeps only a handful of models, our L1 can afford the re-ranking side: model tokens play the role of item features, request tokens play the role of the query, and the joint encoding lets the two sides interact.

A shared linear head scores the hidden state of every model token. Writing $h_m \in \mathbb{R}^d$ for the hidden state at the position of the model token a_m, the head produces

$$z_m = w^\top h_m, \qquad p_m = \sigma(z_m), \tag{2}$$

with $w \in \mathbb{R}^d$. Our L1 is trained with two complementary terms. The first is a per-model binary cross-entropy

$$\mathcal{L}_{\mathrm{bce}} = -\frac{1}{NM}\sum_{i=1}^{N}\sum_{m}\left[y_{i,m}\log p_{i,m} + (1-y_{i,m})\log(1-p_{i,m})\right], \tag{3}$$

which fits every model's own probability of answering the request correctly. The binary cross-entropy fits each model independently and never compares the models with one another. However, the deployment decision is a comparison: L1 picks the model with the highest score, so the decision is correct exactly when the top-scored model is one that answers this request correctly. What matters, then, is how the models rank against one another on each request, not each model's absolute probability. We therefore add the listwise ranking loss of ListNet (Cao et al., 2007),

$$\mathcal{L}_{\mathrm{list}} = -\frac{1}{|\mathcal{P}|}\sum_{i\in\mathcal{P}}\sum_{m} t_{i,m}\log\frac{\exp(z_{i,m})}{\sum_j \exp(z_{i,j})}, \qquad t_{i,m} = \frac{y_{i,m}}{\sum_j y_{i,j}}, \tag{4}$$

where $\mathcal{P} = \{\, i : \sum_m y_{i,m} > 0 \,\}$ collects the requests that at least one model answers correctly. The target t_i spreads the probability mass uniformly over the models that answer x_i correctly. Unlike

the per-model BCE, this term scores the whole pool on each request, lifting the models that answer x_i correctly above those that fail. Requests outside $\mathcal{P}$ contribute to $\mathcal{L}_{\text{bce}}$ only. The total loss is $\mathcal{L} = \mathcal{L}_{\text{bce}} + \mathcal{L}_{\text{list}}$.

At inference L1 routes the request to

$$m^*(x) = \arg\max_m z_m(x) \tag{5}$$

in one forward pass. Pool changes remain cheap. Retiring a model requires no retraining: masking its model token removes it from the decision. Adding a model requires its correctness record on the same K requests and one light fine-tuning run of the same encoder. We study pools that change over time in Section 4.

3.2 L2: A Calibrated Probe Gate

L2 decides whether the routed model suffices or the flagship should take over. It reads the selected model's own prefill hidden states. For every model m we fit a probe on these states, as in prefill probes for correctness prediction (Moreno Cencerrado et al., 2026; Lugoloobi et al., 2026): the last prompt token's hidden state, at a layer chosen by cross-validated AUROC (Fawcett, 2006), reduced with PCA (Pearson, 1901) to 128 dimensions and classified by a logistic regression (Cox, 1958). The probe is fit per model, since hidden states differ across models, and on the pooled training requests of all domains, so the gate needs no domain label at deployment. Probe scores are calibrated into probabilities with out-of-fold predictions and isotonic regression (Zadrozny & Elkan, 2002; Guo et al., 2017). We write $\tilde{v}_m(x)$ for the calibrated probability that model m answers x correctly. At inference, only the probe of the routed model is evaluated. Its input comes for free, since serving would prefill the request anyway and the probe reads the hidden states that this prefill already produces.

The gate weighs the accuracy that an escalation buys against the cost it adds. Let $u(x)$ be the number of input tokens of x, let $\bar{\ell}_m$ be the average output length of model m on the training data, and let c_m^{in} and c_m^{out} be its per-token input and output prices. The relative price ratio is

$$\rho_m(x) = \frac{u(x)\, c_m^{\text{in}} + \bar{\ell}_m\, c_m^{\text{out}}}{u(x)\, c_F^{\text{in}} + \bar{\ell}_F\, c_F^{\text{out}}} > 0, \tag{6}$$

the estimated cost of serving x with model m over the estimated cost of serving it with the flagship. Both estimates use the request's own input length and training-set average output lengths, so the gate decides from information available before inference. We escalate exactly when the estimated accuracy gain of escalation exceeds λ times this ratio,

$$1 - \tilde{v}_{m^*}(x) > \lambda\, \rho_{m^*}(x), \tag{7}$$

where the gain is the routed model's estimated failure probability and 1 on the left-hand side assumes the flagship answers correctly. The parameter $\lambda \geq 0$ is the user's cost preference: it ranges from always escalating, at $\lambda = 0$, to never escalating, as it grows.

The whole pipeline requires only that the selected model's probe can be read, so the flagship can be any stronger model, including an API-based one. For the same reason, the architecture accommodates existing routers: any router that selects among the small models can serve as the first stage, with L2 deciding escalation on top of it.

4 Experiments

We evaluate RTP on a benchmark that mixes eleven public datasets, served by a pool of seven self-deployed small models and an API-based flagship. Section 4.1 compares RTP with existing routing and cascading methods under their native policies, Section 4.2 studies several routers inside our architecture, and Section 4.3 examines pool changes and the latency routing adds to serving.

Data. The benchmark mixes eleven public datasets: TriviaQA, PopQA, MMLU-Pro, BoolQ, CommonsenseQA, ARC, GSM8K, MATH, LogiQA2, IFEval, and LiveCodeBench, covering factual question answering, commonsense, mathematical and logical reasoning, instruction following, and

code generation, with both objective and open-ended answer formats. Table 4 lists each domain with its task, answer format, and split sizes. Due to space limitations, we place it in Appendix A.

Model pool. The pool contains seven self-deployed open-weight models of 7–9B parameters, qwen3.5-9b, llama3-8b, mistral-7b, qwen2.5-7b, qwen2.5-math-7b, gemma-2-9b, and qwen2.5-coder-7b, spanning general-purpose models as well as math and code specialists. The flagship is qwen3.7-max, an API model. Table 5 lists each model with its parameters, specialty, price, and the prefill layer read by its probe, provided in Appendix A.

Baselines. Section 4.1 compares RTP with the pool routers Avengers-Pro, PrefillRouter, kNN-Router, IRT-Router, and CascadeRouting, and the two-model methods RouteLLM, PIKA (cascade form), and FrugalGPT. Section 4.2 additionally compares EmbedLLM, Model-SAT, and ICL-Router, which have no native cost parameter. All of them are run inside our architecture instead. Appendix B lists the encoder, training objective, cost parameter, route cost overhead, and deployment details of every method.

Metric. Our main metric is the average performance gap recovered (APGR) (Ong et al., 2025). APGR measures how much of the accuracy gap between a weak and a strong model that a method closes at each cost budget. Running a method at different settings of its cost parameter yields operating points (cost, accuracy). For a budget B, let $A(B)$ be the highest accuracy among its operating points of cost at most B, interpolating linearly between adjacent points. APGR averages the recovered fraction over a common set of budgets G,

$$\mathrm{APGR} = \frac{1}{|G|} \sum_{B \in G} \frac{A(B) - A_{\mathrm{weak}}}{A_{\mathrm{strong}} - A_{\mathrm{weak}}}, \tag{8}$$

where 1 means flagship quality at every budget and 0 means always calling the weak anchor. The anchors are set per experiment: in Sections 4.1 and 4.2 the weak anchor is qwen3.5-9b, the strongest single small model, and the strong anchor is the flagship, qwen3.7-max. Costs are accounted per query at the real recorded input and output token counts, including each method's own routing overhead. Appendix C gives the budget set and the interpolation details.

4.1 Cost-Accuracy Comparison

All compared pool routers route directly over the mixed pool of the seven small models and the flagship. RouteLLM, PIKA, and FrugalGPT choose between one small model and one strong model, and for a fair comparison we reproduce them with the choice between qwen3.5-9b, the strongest small model in our pool, and the flagship. PIKA's native pool-routing rule requires a correctness probe for each candidate; since our flagship is accessible only through an API, we evaluate its small-to-large cascade variant in the main comparison: the probe of qwen3.5-9b triggers escalation at a threshold.

Figure 3 shows the accuracy of every method at each cost level, and Table 1 summarizes the same comparison with APGR. RTP recovers about 64.6% of the accuracy gap on average and outperforms the best baseline, Avengers-Pro, by 4.4 points. The gains come in two forms. In the low-budget region, where cost is the priority, our margin reaches its largest value over the cascade methods, whose strongest member is PIKA, because routing inside the small pool recovers complementarity that a single fixed small model cannot reach. As the budget grows and accuracy becomes the priority, RTP rises quickly and stays above every baseline across the evaluated range, because the calibrated probe escalates only the requests the routed model is likely to fail, so the flagship is spent more precisely than under mixed-pool routing.

4.2 Component Analysis

EmbedLLM, Model-SAT, and ICL-Router have no native cost parameter. We therefore run all of them inside our architecture: each routes among the seven small models, and our L2 probe gate decides escalation. Table 2 reports the results, with our L1 and the domain oracle for reference.

Our L1 attains the highest APGR among the deployable methods. The domain oracle is not deployable, since it reads the ground-truth domain label at inference and picks the model with the best training accuracy on that domain. L1 is the closest to it among all deployable routers, without using

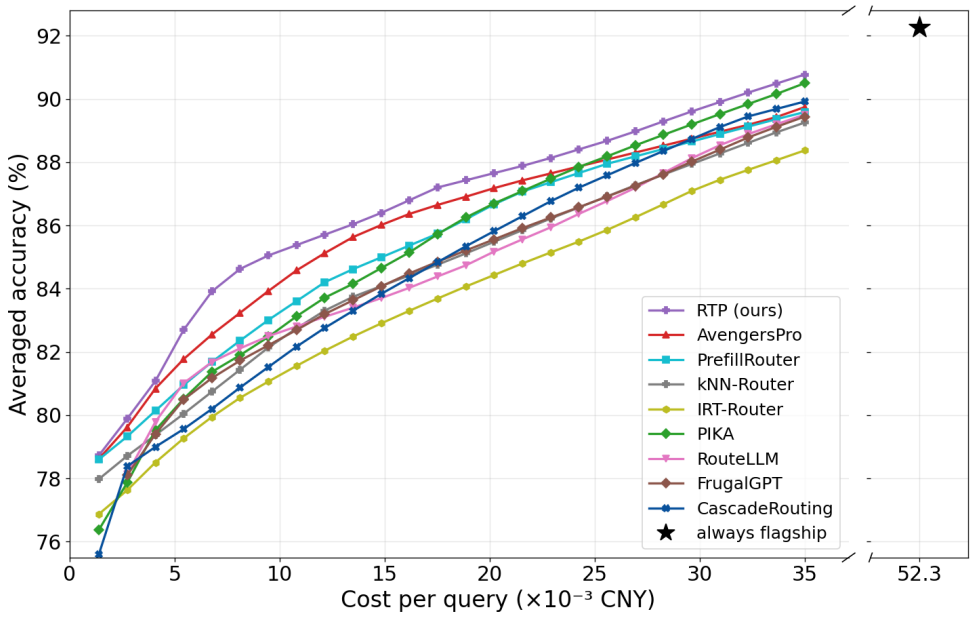


Figure 3: Cost-accuracy curves of all methods. Each curve is obtained by sweeping the method's cost parameter, and the star marks always calling the flagship.

Table 1: Main results. APGR over the common budget region, with the best value in bold.

Method	APGR
RTP (ours)	**0.6463**
Avengers-Pro	0.6026
PrefillRouter	0.5696
PIKA	0.5637
CascadeRouting	0.5222
FrugalGPT	0.5167
kNN-Router	0.5152
RouteLLM	0.5148
IRT-Router	0.4514

a domain label at any stage. Compared to the native policies in Table 1, our two-stage cascade architecture improves every pool router: PrefillRouter gains 6.8, kNN-Router 10.9, IRT-Router 14.5, and Avengers-Pro 1.8 points of APGR. A native policy decides routing and escalation from the same request-side scores, which can hardly tell whether a model will actually answer a request correctly. Its cost knob only sets how often the flagship is called, so the flagship is spent less precisely than by the calibrated probe, which reads the routed model's own prefill hidden states and escalates only the requests it is likely to fail.

4.3 Pool Changes and Latency

The model pool of a serving system changes over time. Table 3 tracks thirteen pool states: stages A1–A7 grow the pool from the weakest model alone to all seven, and stages R6–R1 then retire models from the strongest down. Each stage re-anchors APGR to its own strongest small model, whose test accuracy is listed per model in Table 5. Adding a model retrains L1 and fits the new model's probe. The 149M-parameter L1 encoder retrains quickly, and the prefill hidden states needed for the probe are already produced while generating the new model's training labels. Retiring a model requires no retraining, since masking its model token removes it from the routing decision. RTP has the best cross-stage mean (0.5026) and wins 10 of the 13 stages.

Table 2: Ablation: every method routes among the seven small models first and then escalates through our L2. The last four columns are APGR restricted to budgets of at most 10%, 20%, 30%, and 50% of the flagship's per-query cost. Bold marks the best value in each column; the second best is underlined. The domain oracle uses ground-truth domain labels and is not deployable.

Method	All	$\leq 10\%$	$\leq 20\%$	$\leq 30\%$	$\leq 50\%$
Domain oracle	**0.6516**	**0.2670**	**0.4052**	**0.4804**	**0.5833**
RTP (ours)	<u>0.6463</u>	<u>0.2558</u>	<u>0.3952</u>	<u>0.4706</u>	<u>0.5745</u>
PrefillRouter	0.6377	0.2528	0.3874	0.4617	0.5665
kNN-Router	0.6241	0.2192	0.3642	0.4442	0.5519
Avengers-Pro	0.6209	0.2389	0.3693	0.4421	0.5477
EmbedLLM	0.6173	0.1963	0.3418	0.4241	0.5404
IRT-Router	0.5967	0.1475	0.3034	0.3921	0.5156
Model-SAT	0.5878	0.1392	0.2920	0.3831	0.5070
ICL-Router	0.5315	0.0195	0.1480	0.2671	0.4306

Routing must also not slow first-token latency. Due to space limitations, the detailed latency measurements are in Appendix D. On average, our L1 adds 5.0 ms, and L2 evaluating the probe costs 0.2 ms per request. A request that stays on the routed small model only pays these 5.2 ms, against the small-model prefill of about 22 ms that serving pays anyway. A request that escalates to the flagship additionally wastes that prefill, so its routing overhead reaches about 28 ms on average, still very small against the roughly 700 ms TTFT of the flagship API call.

Table 3: Pool changes. Each stage updates the pool and re-evaluates every method with anchors set to the stage's strongest small model versus the flagship. A1–A7 add models from the weakest to the strongest; R6–R1 remove models from the strongest down. The last row is the cross-stage mean. Bold marks the best value in each row.

Stage	RTP (ours)	PrefillRouter	kNN-Router	CascadeRouting	IRT-Router	Avengers-Pro
A1	0.3773	**0.3952**	0.3589	0.3284	0.3931	0.3312
A2	**0.5369**	0.4720	0.4366	0.4400	0.4557	0.5266
A3	**0.5243**	0.4320	0.4024	0.4165	0.4079	0.4653
A4	**0.5506**	0.4643	0.4249	0.4084	0.4196	0.4737
A5	**0.5110**	0.4340	0.3674	0.3909	0.3143	0.4305
A6	**0.4763**	0.4120	0.3715	0.3752	0.3790	0.4225
A7	**0.6463**	0.5696	0.5152	0.5222	0.4514	0.6026
R6	**0.4872**	0.3893	0.3715	0.3752	0.2872	0.4225
R5	**0.5136**	0.4192	0.3674	0.3909	0.2858	0.4305
R4	**0.5522**	0.4588	0.4249	0.4084	0.3528	0.4737
R3	**0.4962**	0.4345	0.4024	0.4165	0.3496	0.4653
R2	0.4842	0.4674	0.4366	0.4400	0.4318	**0.5266**
R1	0.3773	**0.3859**	0.3589	0.3284	0.3650	0.3312
Mean	**0.5026**	0.4411	0.4030	0.4032	0.3764	0.4540

5 Conclusion

In this paper, we study how to make cost-effective use of request-side and inference-time signals for LLM routing, and propose Route-then-Probe (RTP). Its L1 router jointly encodes the request and the features of all candidate small models in one forward pass and selects the best-suited one, and its L2 gate escalates with a calibrated probe on the selected model's prefill hidden states. On a mixed benchmark spanning eleven domains, RTP recovers about 64.6% of the accuracy gap between the strongest small model and the flagship, outperforming the best baseline by 4.4 points of APGR. Our architecture can incorporate and improve existing routers, its routing overhead is small, and it remains robust as the model pool changes.

AI USE STATEMENT

In this work, we used generative AI tools for identifying relevant literature, for drafting parts of the experimental code, and for editing the manuscript to improve readability. The research ideas, the study design, the experimental setup, and the interpretation of the results are our own. We have not used generative AI tools for generating datasets, developing mathematical claims, or assisting with proofs, and these tasks are not applicable to this work. We have reviewed all AI-assisted work: AI-suggested related work was verified against the original papers and cited accordingly, AI-assisted code was reviewed by the authors and all reported numbers are produced by the reviewed code, and AI-polished text was checked for accuracy of every technical claim. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All experiments in this paper use public datasets and open-weight models. Our experimental code will be submitted as Supplementary Material.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. AutoMix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, 2024.

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 129–136, 2007.

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*, 2024.

Zhou Chen, Zhiqiang Wei, Yuqi Bai, Xue Xiong, and Jianmin Wu. TagRouter: Learning route to LLMs through tags for open-domain text generation tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. 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)*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

David R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for LLMs. In *International Conference on Machine Learning*, 2025.

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.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

Aashna Garg, Siddharth Singha Roy, Jinu Jang, Federico Brancasi, Giuseppe Cianci, and Shengyu Fu. HyDRA: Hybrid dynamic routing architecture for heterogeneous LLM pools. *arXiv preprint arXiv:2605.17106*, 2026.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, et al. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1321–1330, 2017.

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 *Advances in Neural Information Processing Systems*, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Upadhyay. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *International Conference on Learning Representations*, 2020.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Hanmeng Liu, Jian Liu, Leyang Cui, et al. LogiQA 2.0—an improved dataset for logical reasoning in natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

William Lugoloobi, Thomas Foster, William Bankes, and Chris Russell. LLMs encode their failures: Predicting success from pre-generation activations. In *ICLR 2026 LIT Workshop*, 2026.

Alex Mallen, Akari Asai, Victor Zhong, et al. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

Iván Vicente Moreno Cencerrado, Arnau Padrés Masdemont, Anton Gonzalez Hawthorne, David Demitri Africa, and Lorenzo Pacchiardi. No answer needed: Predicting LLM answer accuracy from question-only linear probes. In *ICLR 2026 Workshop on Principled Design for Trustworthy AI*, 2026.

Yasmin Moslem and John D. Kelleher. Dynamic model routing and cascading for efficient LLM inference: A survey. *Transactions on Machine Learning Research*, 2026.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020.

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.

Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. CARROT: A cost aware rate optimal router. *arXiv preprint arXiv:2502.03261*, 2025.

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)*, 2025.

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)*, 2019.

Tanay Varshney, Annie Surla, Michelle Xu, Gomathy Venkata Krishnan, Maximilian Jeblick, David Austin, Neal Vaidya, and Davide Onofrio. LLM router: Rethinking routing with prefill activations. *arXiv preprint arXiv:2603.20895*, 2026.

Chenxu Wang, Hao Li, Yiqun Zhang, Linyao Chen, Jianhao Chen, Ping Jian, Peng Ye, Qiaosheng Zhang, and Shuyue Hu. ICL-Router: In-context learned model representations for LLM routing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 105–114, 2011.

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)*, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

Cheng Yan, Wuyang Zhang, Zhiyuan Ning, Fan Xu, Ziyang Tao, Lu Zhang, Bing Yin, and Yanyong Zhang. Cost-efficient zero-shot LLM routing via a universal latent space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 694–699, 2002.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. In *Conference on Language Modeling*, 2025a.

Yi-Kai Zhang, De-Chuan Zhan, and Han-Jia Ye. Capability instruction tuning: A new paradigm for dynamic LLM routing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025b.

Yiqun Zhang, Hao Li, Jianhao Chen, Hangfan Zhang, Peng Ye, Lei Bai, and Shuyue Hu. Beyond GPT-5: Making LLMs cheaper and better via performance-efficiency optimized routing. In *International Conference on Distributed Artificial Intelligence*, 2025c.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, et al. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

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.

A Data, Models, and Splits

Table 4: The eleven domains. Size is the number of candidate questions per domain, from which our splits are sampled; Train and Test give the split sizes.

Domain	Task	Answer format	Size	Train	Test
TriviaQA (Joshi et al., 2017)	Factual QA	Open-ended	9958	2000	600
PopQA (Mallen et al., 2023)	Factual QA	Open-ended	14267	2000	500
MMLU-Pro (Wang et al., 2024)	Knowledge	Multiple-choice	12032	2000	497
BoolQ (Clark et al., 2019)	Reading comp.	Yes/no	9427	2000	500
CommonsenseQA (Talmor et al., 2019)	Commonsense	Multiple-choice	9741	2000	500
ARC (Clark et al., 2018)	Science reasoning	Multiple-choice	2291	1000	500
GSM8K (Cobbe et al., 2021)	Grade-school math	Open-ended	8792	2000	499
MATH (Hendrycks et al., 2021)	Competition math	Open-ended	8000	2000	498
LogiQA2 (Liu et al., 2023)	Logical reasoning	Multiple-choice	12567	2000	489
IFEval (Zhou et al., 2023)	Instruction following	Open-ended	541	240	301
LiveCodeBench (Jain et al., 2025)	Code generation	Open-ended	1055	555	399
Total				17795	5283

Table 5: Model pool and prices in CNY per million input and output tokens. The seven small models share one price schedule. Probe layer is the prefill layer the L2 probe reads for each model. Checkpoints: Qwen3.5-9B, Llama-3-8B-Instruct, Mistral-7B-Instruct-v0.3, Qwen2.5-7B-Instruct, Qwen2.5-Math-7B-Instruct, gemma-2-9b-it, and Qwen2.5-Coder-7B-Instruct.

Model	Parameters	Specialty	Probe layer	Test acc	Input	Output
qwen3.5-9b	9B	General-purpose	17	0.7511	0.5	2.0
llama3-8b	8B	General-purpose	18	0.5737	0.5	2.0
mistral-7b	7B	General-purpose	29	0.4932	0.5	2.0
qwen2.5-7b	7B	General-purpose	24	0.6477	0.5	2.0
qwen2.5-math-7b	7B	Math specialist	19	0.4703	0.5	2.0
gemma-2-9b	9B	General-purpose	26	0.6342	0.5	2.0
qwen2.5-coder-7b	7B	Code specialist	20	0.5856	0.5	2.0
qwen3.7-max (API)	–	Flagship	–	0.9226	12	36

B Baselines and Implementation Details

Table 6 lists the encoder, training objective, cost parameter, and route tax of every method. Route taxes are charged per query: routers built on an embedding model pay its input tokens at the small-model price, the BERT-style routers add effectively zero, and the routers that run a judge LLM pay a fixed per-query fee for the judge's inference. The last three methods in the table have no native cost parameter, so they appear only in the component analysis, where each routes among the small models and our L2 decides escalation.

Deployment of each method. Our L1 averages the per-arm probabilities of five training seeds for every request; L2 is a single calibrated probe per model. The trained baselines that involve randomness follow their own conventions on top of the same five seeds. PrefillRouter follows its original protocol and keeps the seed with the best validation AUROC; IRT-Router, Avengers-Pro, RouteLLM, and FrugalGPT average their scores over the five seeds. kNN-Router, CascadeRouting, and PIKA are deterministic in our reproduction: they are either training-free or fitted with deterministic procedures, and PIKA additionally reuses our calibrated probe scores.

Our L1 is trained with AdamW, weight decay 0.01, a cosine schedule with 10% warmup, 10 epochs, learning rate 2×10^{-5}, batch size 32, and maximum sequence length 1024. The fingerprint anchors are sampled from the training questions only ($K = 216$). The L2 probes read the hidden state of the last prompt token, scanning all layers; for each model the layer is selected by 5-fold cross-validated

AUROC on the training questions (Table 5 lists the selected layer), and the state is reduced by PCA to 128 dimensions and classified by a logistic regression; probe scores are calibrated with 5-fold out-of-fold predictions and isotonic regression, and the probes are fitted on the training questions of all domains jointly. All reported numbers are computed on the test questions. kNN-Router uses $k = 80$ neighbors, selected on the validation split.

C THE APGR METRIC

We evaluate all methods on the same set of budgets G over the common region $[1.41, 35] \times 10^{-3}$ CNY per query. The weak anchor (the strongest single small model) costs 1.40×10^{-3} CNY per query and the strong anchor (the flagship) 52.31×10^{-3}. Below a method's cheapest operating point it is infeasible, and beyond its most expensive point $A(B)$ stays at the method's top accuracy. Between two adjacent Pareto points we interpolate linearly, since mixing the two operating points by a probability realizes any point on the segment.

D LATENCY MEASUREMENT

All latency numbers are reported as the added TTFT relative to serving without routing, for the production configuration: L1 is the five-seed ensemble with members run in parallel, one member per GPU. All GPU-side components are measured on RTX PRO 5000 (72GB) GPUs with the small models served by vLLM 0.26 at batch size 1 with cold prefill, meaning the key-value cache is empty when a request arrives; the probe runs on CPU (AMD EPYC 9A75). Per-query decisions are reconstructed with the production decision rule on all 5283 test questions. Component latencies, averaged over the request length distribution: the parallel L1 ensemble adds 5.0 ms, the PCA + LR probe 0.22 ms on CPU, and the routed model's prefill 22.0 ms; the flagship API returns its first token after roughly 700 ms. Table 7 reports the added TTFT at several escalation budgets, sweeping the escalation parameter λ; the escalation rate is measured on the test set.

Table 6: Encoders, training objectives, cost parameters, and route taxes of all methods. Route taxes are the per-query cost of running the router itself.

Method	Encoder and features	Training objective	Cost parameter	Route tax
Ours L1	ModernBERT-base (149M); [CLS] + 7 model tokens + request; one forward pass	BCE + ListNet	λ (via L2)	≈ 0
Ours L2	per-model probe on prefill hidden states; PCA 128 + LR, OOF + isotonic	calibrated BCE	λ	prefill already paid
PrefillRouter (Varshney et al., 2026)	qwen3.5-9b prefill hidden states; Fisher layer selection, PCA 256 + 2-layer MLP	BCE	$\lambda p - (1-\lambda)\hat{c}$	qwen3.5-9b prefill input
Avengers-Pro (Zhang et al., 2025c)	Qwen3-Embedding-8B; 60 clusters and profiles	cluster-level regression	$\alpha \cdot \hat{c}$	8B embedding input
kNN-Router (Shnitzer et al., 2023)	cached embeddings; $k = 80$ neighbors	none	$\hat{p} - \alpha \cdot \hat{c}$	≈ 0
IRT-Router (Song et al., 2025)	frozen ModernBERT CLS; 25-dim abilities	BCE	$\lambda \cdot \Delta p - \hat{c}$	≈ 0
RouteLLM (Ong et al., 2025)	fine-tuned ModernBERT; binary "needs flagship"	BCE	threshold	≈ 0
FrugalGPT (Chen et al., 2024)	qwen3.5-9b generation; ModernBERT judge	judge BCE	scorer threshold	qwen3.5-9b generation
PIKA (Lugoloobi et al., 2026)	qwen3.5-9b prefill hidden states; last-token linear probe	probe LR	threshold	qwen3.5-9b prefill input
CascadeRouting (Dekoninck et al., 2025)	per-model LR quality classifiers on 8B embeddings; logprob-based post-hoc deferral	LR + isotonic	λ	classifier + deferral generation
EmbedLLM (Zhuang et al., 2025)	MPNet-base (110M); model embeddings from the correctness matrix	BCE	none	≈ 0
Model-SAT (Zhang et al., 2025b)	E5-large profiles (frozen); Phi-3-mini judge with soft prefix	judge CE	none	flat 1.6×10^{-3} CNY
ICL-Router (Wang et al., 2026)	Qwen3-Embedding-8B profiles; Qwen2.5-7B judge with soft prefix	judge CE	none	flat 2.7×10^{-3} CNY

Table 7: Added TTFT (ms) at several operating points. λ is the escalation parameter; non-escalating requests pay the parallel L1 ensemble and the probe, and escalating requests additionally pay the wasted small-model prefill.

λ	Escalation rate	Mean	$p50$	$p95$	$p99$
5.4	50.5%	16.9	19.6	46.0	71.1
8.6	41.4%	14.8	5.8	44.3	70.5
13.6	30.5%	11.9	5.2	28.7	65.5
21.5	20.5%	9.3	4.9	24.1	52.2
34.2	14.9%	7.7	4.9	20.2	24.1
54.1	10.2%	6.9	4.8	20.2	24.1
100	4.3%	5.9	4.8	11.2	20.2