LENS-ROUTER: PROGRESSIVE FINE-GRAINED ROUTING FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

The LLM ecosystem is undergoing rapid model updates and intense tiered competition, requiring LLM routers to support efficient model expansion and fine-grained routing. However, existing routing paradigms face a dilemma between scalability and fine-grained routing: finer-grained decisions require more historical supervision, making new model incorporation less scalable. Motivated by this dilemma, we propose Lens-Router, a progressive fine-grained router that abstracts reusable, cross-model resource demand features from queries. These features provide effective supervision for both capability filtering and realized cost prediction, enabling Lens-Router to efficiently incorporate new models and select the lowest realized cost model that can fulfill the query. Our experiments show that with only 700 historical supervision for model adaptation, Lens-Router improves routing accuracy while reducing realized cost by up to 82.8% compared with SOTA LLM routers.[1]

1 INTRODUCTION

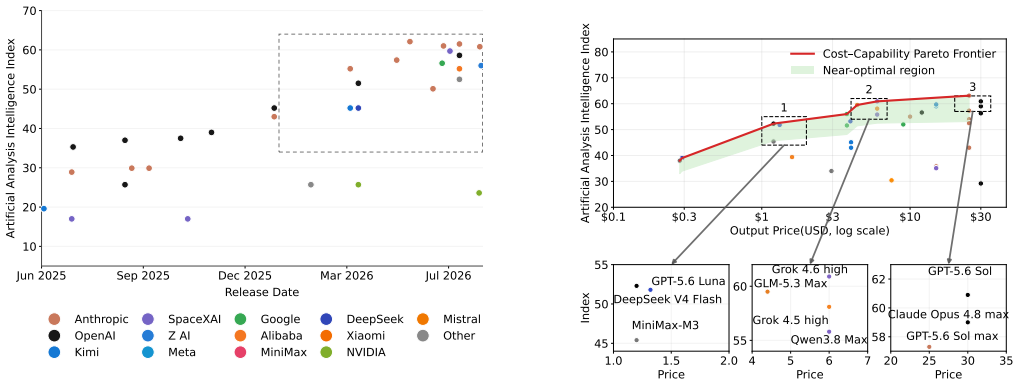
LLM routers enable efficient LLM serving by balancing task quality and cost, predicting how candidate models perform on a query and selecting the most suitable one (Hu et al., 2024). However, the rapid evolution of the LLM ecosystem is reshaping this matching problem. As shown in Figure 1, model pools are continuously changing as older models are quickly replaced (Demirer et al., 2026). At the same time, competition has pushed models into dense clusters at several points along the cost-capability Pareto frontier, where models within each cluster are priced comparably and appear similarly capable on average. This similarity makes capability boundaries within each cluster difficult to distinguish, while comparable prices among these models can mask substantial differences in their realized costs. An effective router therefore needs to incorporate new models with limited supervision, while distinguishing both clearly different models and apparently similar models.

Existing LLM routers primarily use each model's observed outcomes across a range of queries as the basis for routing decisions (Ding et al., 2024; Ong et al., 2025). This paradigm inherently benefits from broader query coverage, as more observations provide better evidence for matching queries with the most suitable models. While effective, it fails to address the new challenges posed by the evolving LLM ecosystem. As model pools become denser, fine-grained routing must capture factors beyond capability and increasingly subtle differences between models, naturally requiring broader query coverage. However, broader historical supervision increases the cost of incorporating new models, while using less supervision inevitably sacrifices part of the router's discrimination capability.

This dilemma motivates us to propose Lens-Router, a progressive fine-grained router for large language models. Lens-Router is designed to select a model that minimizes realized cost without sacrificing answer quality under limited historical supervision. Our key observation is that although the outcomes of a query remain model-dependent, its relative resource demands are largely shared across models. For example, solving a complex mathematical problem requires greater reasoning capability and realized cost than merely explaining what the problem asks. These resource demand features closely align with the information required for routing, providing effective and reusable prior knowledge that allows the router to avoid learning directly over the large query space. Therefore we first train a resource demand encoder on cross provider historical responses to capture the

[1] Our code and models will be released after acceptance.

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.



(a) Frontier LLM releases have become increasingly dense, with models from multiple providers emerging in rapid succession.

(b) Intensifying market competition narrows capability gaps and drives prices toward concentrated tiers, giving rise to dense clusters of models with similar cost and capability.

Figure 1: Emerging Trends in the LLM Ecosystem.

capability and cost demands of each query. Based on this representation, we group models into capability clusters and predict the minimum capability required by each query. We then characterize the model-specific cost expansion patterns and learn pairwise relative cost rankings among similarly priced models, allowing the router to capture query-conditioned differences in realized cost. Through progressively filtering models by capability and ranking the remaining candidates by realized cost, Lens-Router selects the lowest-cost model that satisfies the query's capability requirement while reducing the need for model-specific historical supervision.

Our experiments show that across nine benchmarks and a diverse pool of newly released models, Lens-Router consistently achieves higher accuracy than the strongest baseline, and generally reduces total realized cost with savings of up to 82.8%. In a model pool deliberately constructed from closely matched capability and price clusters, Lens-Router further raises average accuracy over strongest baseline while reducing cost on all benchmarks, demonstrating its advantage in fine-grained routing among apparently similar models. Notably, Lens-Router requires only 700 historical queries for each model to achieve these results. Our ablation studies further validate the effectiveness of each component of Lens-Router. In summary, our contributions are:

- We reformulate LLM routing by introducing query resource demand as a reusable signal for model selection beyond semantic content.
- We introduce a progressive router that decomposes model preference prediction with progressively finer decision, addressing the challenges posed by rapidly evolving and increasingly clustered LLMs.
- Extensive experiments demonstrate the effectiveness and scalability of Lens-Router across diverse benchmarks and model pools.

2 Related Work

LLM routing aims to select an appropriate model for a given query before the model generates a response, requiring the system to predict relevant characteristics of candidate models' responses and translate these prediction into query–model matching decisions. Based on this perspective, we organize related work into two lines: predicting characteristics of model responses and developing mechanisms for LLM routing.

Model Response Characteristic Prediction. Predicting characteristics of model responses can provide useful signals not only for estimating realized cost, but also for resource scheduling, adaptive computation, and efficient service management. Some work predict response or output length

from input queries and use the estimates to improve sequence scheduling and task grouping (Zheng et al., 2023; Xia et al., 2024; Fu et al., 2026). Other studies exploit intermediate representations of the target model to estimate remaining generation length, enabling more fine-grained inference scheduling (Piotrowski et al., 2025). Beyond resource-related characteristics, model internal representations have also been shown to contain information about future response behaviors (Ashok & May, 2026). Recent work further extends response prediction to multi-cloud workload scheduling and execution-time estimation, demonstrating its potential for resource allocation and adaptive inference (Cong et al., 2026; Fan et al., 2026). However, existing approaches largely target individual models or local serving and rely on model-specific information unavailable for many proprietary, API-accessed models.

LLM Router. Existing LLM routers largely decouple query-side and model-side information. One line of work assumes that the query contains sufficient information for routing and directly learns a mapping from queries to suitable models. Learning-based routers formulate model selection as a supervised prediction problem using historical supervision data (Ding et al., 2024; Ong et al., 2025; Zhang et al., 2025a; Wang et al., 2026b). Another line instead assumes that each model can be adequately represented by a static vector learned from its behavior over representative queries. These works (Zhuang et al., 2025; Wang et al., 2026a; Jitkrittum et al., 2026)learn compact model representations from performance or response profiles and use them for routing and related downstream tasks. While these routers improve routing scalability, they either emphasize query representations or treat model characteristics as static, overlooking the query-conditioned variation in model capabilities and resource demands.

3 METHODOLOGY

Lens-Router is designed to select a model that minimizes realized cost without sacrificing answer quality under limited historical queries. Lens-Router decomposes routing into three stages, as shown in Figure 2. 1) We first train a resource demand encoder that maps each query to a reusable representation of its capability and resource requirements. 2) Based on this representation, we group models into capability clusters and predict the minimum capability level required by each query. 3) We then characterize model-specific cost expansion patterns and learn pairwise relative cost rankings among models with similar prices, capturing query-conditioned differences in realized cost. Through this progressive process, Lens-Router first filters models by capability and then ranks the remaining candidates by realized cost, selecting the lowest-cost model that satisfies the query's capability requirement while reducing the need for model-specific historical supervision.

Resource Demand Encoder To achieve both accurate and scalable routing, we seek a query representation that captures reusable information shared across models rather than model-specific outcomes. We observe that the relative resource demands induced by different queries are largely stable across models, providing a common signal that can be reused as the model pool evolves, as empirically demonstrated in Appendix A.2. This signal is also more closely aligned with routing decisions than generic semantic representations. For example, ''What is Fermat's Last Theorem?' and ''Prove Fermat's Last Theorem' are semantically similar, yet the latter requires substantially greater reasoning capability and inference cost. A semantic representation may treat the two queries as highly similar, while a resource demand representation explicitly captures the difference in what the models need to fulfill them. We thus abstract from each query a reusable, cross-model representation of its capability and resource requirements, which we define as **resource demand**. As formally established in Appendix A.3, routing based on this representation requires fewer model-specific samples under the stated assumptions.

To develop reliable resource demand representations, we construct a broad pool of candidate resource demand dimensions through literature review (Wankhade & Shahade, 2026; Samuel et al., 2025; Narita et al., 2026) and LLM-assisted analysis of representative queries with substantial variation in answering requirements. The candidate set is informed by factors repeatedly identified in prior work on query complexity and task requirements. We then select 3,000 queries spanning the full difficulty range of the source datasets and execute them on multiple LLMs, score the candidate dimensions based on their observed responses, and perform correlation analysis to remove redundant dimensions, yielding 24 relatively independent dimensions that broadly characterize query resource demand. The detailed dimension selection process and results are provided in Appendix B.1.1.

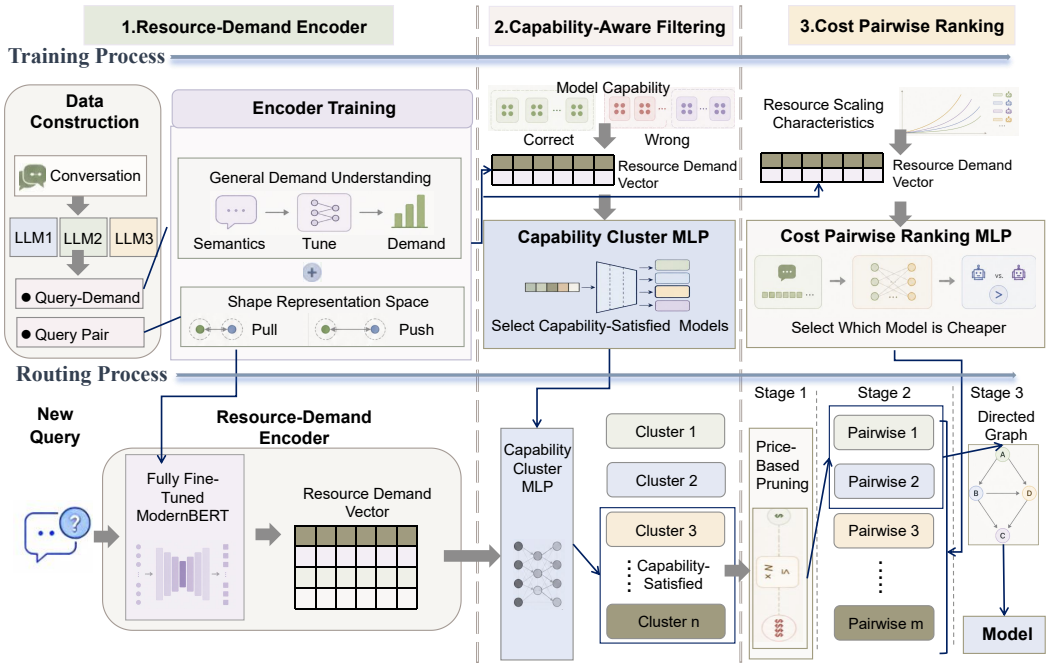


Figure 2: Overview of the proposed query-aware model router. The upper part illustrates the training process. The lower part shows the routing process for a new query to identify the final model.

We fully fine-tune ModernBERT (Warner et al., 2025), a moderately sized pretrained encoder with general-purpose representation capability and useful prior knowledge, to learn the resource demand representation. Our training data are constructed from 59,577 distinct queries collected from real-user data in WildChat (Zhao et al., 2024) and the Brick Complexity Extractor (Team, 2026). For each query, we first run it on DeepSeek-V4-Pro, GPT-5.6-Sol, and Claude Opus 4.8 to collect their responses and resource consumption. We provide each LLM with the query, execution results as behavioral evidence, and predefined criteria to independently score all resource demand dimensions; the evaluation prompt is given in Appendix C. We take the arithmetic mean of the three models' scores for each dimension to obtain the final resource demand label. Our validation experiments further demonstrate that these training data provide a reliable and behaviorally meaningful description of resource demand in Appendix B.1.2. We then combine these labels with the semantic representations produced by the original ModernBERT to construct hard query pairs, including semantically similar queries with different demands, long queries with low demands, short queries with high demands, and semantically dissimilar queries with similar demands.

The encoder is trained in two phases. In Phase 1, we fully fine-tune the pretrained encoder so that its hidden representations capture information predictive of the 24-dimensional resource demand vector. Given a query q, the encoder produces a hidden representation $\mathbf{z}_q$, which is mapped to the predicted resource demand vector $\hat{\mathbf{d}}_q$ through a regression head. We optimize the Smooth L1 regression loss:

$$\mathcal{L}_{\text{reg}} = \text{SmoothL1}\left(\hat{\mathbf{d}}_q, \mathbf{d}_q\right), \tag{1}$$

where $\mathbf{d}_q$ denotes the target resource demand vector. In Phase 2, we further incorporate the hard query pairs to explicitly reshape the representation space according to resource demand relationships. We jointly optimize the regression objective and the pairwise contrastive objective:

$$\mathcal{L}_{\text{stage2}} = \lambda_r \mathcal{L}_{\text{reg}} + \lambda_p \mathcal{L}_{\text{pair}}, \tag{2}$$

where $\mathcal{L}_{\text{pair}}$ encourages queries with similar resource demands to have closer representations while separating those with different demands. The regression objective preserves absolute resource demand information, whereas the pairwise objective captures relative demand relationships. Together, they yield a resource demand-aware query representation for subsequent routing. Detailed training procedures and hyperparameter settings are provided in Appendix B.1.3.

Capability-Aware Cluster Filtering The resource demand representation indicates the capability required by a query, but fine-grained capability discrimination among similar models is inherently difficult and often unreliable. Within a capability cluster, small differences in model performance can overlap across queries, making precise model-level boundaries difficult to establish and requiring redundant evaluations to resolve marginal differences. We group models with similar capabilities into clusters and predict the minimum capability required by each query at the cluster level. Our empirical analysis in Appendix A.1 further verifies the instability of fine-grained capability boundaries.

To construct the capability clusters, we evaluate each candidate model on 700 questions from MMLU, Math, LeetCode, XpertBench (Liu et al., 2026a) and record its correctness on each query, all training and evaluation data are strictly disjoint. Based on our cross-benchmark empirical study, the within-benchmark performance variation of a single model is typically below 2%, motivating this clustering threshold. We then feed these queries into the resource demand encoder and use the resulting demand vectors together with the cluster-level correctness labels to train an MLP. To emphasize capability boundaries, we assign higher weights to queries whose correctness differs between adjacent clusters. Specifically, let $y_{q,c}$ denote whether cluster c answers query q correctly. We define the query weight as

$$w_q = 1 + \alpha \sum_c \mathbb{I}\left[y_{q,c} \neq y_{q,c+1}\right], \tag{3}$$

and optimize the weighted classification objective

$$\mathcal{L}_{\text{cluster}} = \frac{1}{N} \sum_{q=1}^{N} w_q \, \mathcal{L}_{\text{cls}}(q). \tag{4}$$

This encourages the classifier to focus on queries that distinguish neighboring capability levels, yielding a coarse set of capability-compatible candidates for subsequent cost-aware ranking.

Cost-Aware Pairwise Ranking The capability filtering stage identifies models that can satisfy the query, but these capability-compatible models can still incur substantially different realized costs. We use the resource-demand representation to predict realized cost differences among the remaining candidates. Precisely predicting the absolute number of output or reasoning tokens is difficult, as generation is inherently variable and its realized length depends on both the query and the model. Moreover, accurately modeling absolute costs requires extensive historical supervision to cover the large query space. We therefore predict the relative cost ordering between models instead of their absolute costs. Relative ordering can effectively exploit the comparative cost patterns between models while requiring less supervision to learn, enabling accurate query-conditioned cost discrimination with limited historical data.

We exploit these relationships for fine-grained model selection. During training, we reuse the 700 queries from the capability evaluation set and record the output and reasoning tokens generated by each model to obtain its realized cost for each query. We pair models whose official prices differ by no more than $2\times$. For model pairs with a price ratio between $2\times$ and $5\times$, we additionally construct a predictor only when a cost-order reversal is observed in the 700 queries. For each model pair (m_i, m_j), we use the resource demand representation of query q and its realized pairwise cost outcome to train an MLP binary classifier that predicts which model is cheaper, together with its confidence. This pairwise formulation avoids the difficulty of absolute cost prediction and facilitates model-pool expansion, as adding a new model only requires constructing the corresponding pairwise comparisons.

At inference time, given the capability clusters identified in the previous stage, we first prune clearly inefficient candidates: When two models have an official price ratio exceeding 5, empirical logs show that the cheaper model's cost does not exceeds the pricier one's even with longer reasoning traces, so we directly remove the more expensive model. We then apply the corresponding pairwise classifiers to the remaining candidates. Their predictions and confidence scores form weighted directed edges between models, yielding a query-specific cost-preference graph. We combine the predictions from all evaluated model pairs to determine the final selected model. The detailed decision procedure, as well as the resulting model clusters and pair configurations, is provided in Appendix B.2.2.

Table 1: Performance (%) of individual models and LLM routers across different benchmarks.

Model	Math		Knowledge				Code		Other
	LBM	MAR	OMN	MMP	GPD	HLE	SCI	SBV	IFB
Qwen-3.8-27B	74.3	13.7	-36.0	72.1	84.5	14.6	43.6	77.6	64.1
Gemini-3.1-Pro	86.7	34.2	32.0	82.9	94.1	47.0	58.7	80.8	77.1
GPT-5.6-Luna	88.0	35.5	-12.0	79.0	91.1	33.4	51.6	80.2	66.7
DeepSeek-V4-Flash	86.8	29.3	-14.0	75.5	86.7	34.8	40.2	87.3	79.2
Gemini-3.7-Flash	87.1	29.6	26.0	85.6	94.5	47.9	59.8	81.1	76.3
GPT-5.6-Sol	92.3	64.1	20.0	81.1	94.1	42.2	57.4	88.5	72.7
Claude-Opus-4.7	87.9	21.0	27.0	76.3	91.4	42.3	52.3	87.6	58.6
Claude-Opus-4.8	88.3	42.7	29.0	77.8	92.0	48.7	54.4	88.6	62.2
Claude-Fable-5	96.0	54.0	43.0	85.4	93.7	54.4	61.0	92.3	63.5
Avengers-Pro $\alpha = 0.0$	87.2	27.8	26.0	83.4	94.2	47.6	58.3	81.3	68.7
Avengers-Pro $\alpha = 0.53$	88.0	57.1	26.9	81.5	94.1	48.3	58.1	88.2	74.1
Avengers-Pro $\alpha = 0.80$	88.6	57.7	27.0	81.7	94.1	48.5	58.1	88.1	74.8
Router-R2	86.7	28.3	-14.0	74.6	85.3	34.9	40.5	78.7	74.2
ICL-Router	91.1	61.4	17.0	79.4	87.8	41.2	57.4	86.3	71.8
Avengers	89.5	46.9	**29.0**	79.5	93.6	48.1	57.4	**89.7**	72.5
Lens-Router (Ours)	**93.7**	**63.2**	28.0	**84.2**	**95.7**	**51.3**	**58.7**	88.9	**75.7**

4 EVALUATION

In this section, we evaluate the overall effectiveness of Lens-Router and the contribution of its individual components through comprehensive comparison and ablation experiments.

Experiment Setup We evaluate Lens-Router on a diverse set of benchmarks spanning four broad domains: Math, Knowledge, Code, and Other, covering LiveBench-Math (LBM (White et al., 2025)), MATH Arena (MAR (Balunovic et al., 2026)), OmniScience (OMN (Jackson et al., 2025)), MMLU-Pro (MMP (Wang et al., 2024)), GPQA-Diamond(GPD (Rein et al., 2023)), HLE (Phan et al., 2025), SciCode (SCI (Tian et al., 2024)), SWE-bench-Verified (SBV (Jimenez et al., 2024)), and IFBench (IFB (Pyatkin et al., 2026)). These benchmarks cover different task types and difficulty levels, allowing us to assess the routing performance across a broad range of real-world queries rather than relying on a single domain or task format. They also provide strong discrimination among frontier models.

We compare Lens-Router against four representative routing baselines: ICL-Router (Wang et al., 2026a), Router-R2 (Xue et al., 2026), Avengers (Zhang et al., 2026), and Avengers-Pro (Zhang et al., 2025b). ICL-Router and Avengers represent recent model-based routing approaches that learn or exploit model-level representations for model selection. Router-R2 explicitly incorporates model output budgets as an important prediction variable, making it particularly relevant to our resource-aware routing objective. Avengers-Pro further considers the quality-cost trade-off by introducing a parameter α to control its routing preference between answer quality and cost, and has been shown in independent evaluations (Li et al., 2026) to achieve a competitive position close to the Pareto frontier. We use $\alpha = 0$, $\alpha = 0.53$ and 0.8 as representative settings, as the resulting model selections largely fall into these regimes. Together, these baselines cover different routing strategies and provide a representative comparison for evaluating both the effectiveness and cost efficiency of Lens-Router.

Main Results We first evaluate the scalability of Lens-Router by measuring the amount of query supervision required to incorporate a newly introduced model. Lens-Router requires 700 queries to learn the new model's routing behavior, substantially fewer than existing approaches. In comparison, ICL-Router requires approximately 1,000 queries, Router-R2 about 1,936 queries, Avengers-Pro about 5,912 queries, and Avengers about 25,444 queries for model-level profiling.

We then evaluate Lens-Router using a diverse set of models released in 2026 by multiple providers, covering nine benchmarks across four domains: Math, Knowledge, Code, and Other. As shown in Table 1, 2, the individual models exhibit substantial performance differences across benchmarks, with their average accuracy ranging from 45.39% to 71.48%, highlighting the large variation in

Table 2: Total realized cost of individual models and LLM routers across different benchmarks.

Model	Math		Knowledge				Code		Other
	LBM	MAR	OMN	MMP	GPD	HLE	SCI	SBV	IFB
Qwen-3.8-27B	6.624	0.97	52.43	8.97	1.65	84.16	3.36	497.32	15.15
Gemini-3.1-Pro	69.18	0.52	106.03	136.76	14.10	424.60	23.97	393.16	21.64
GPT-5.6-Luna	26.12	2.21	15.00	12.10	1.96	20.52	2.67	20.77	1.79
DeepSeek-V4-Flash	12.14	0.10	119.90	12.27	1.47	20.44	1.75	5.74	1.14
Gemini-3.7-Flash	29.72	0.40	23.55	13.11	1.97	79.36	2.89	515.23	6.51
GPT-5.6-Sol	51.15	2.45	512.46	21.39	14.47	579.17	22.90	575.03	41.46
Claude-Opus-4.7	118.49	4.17	213.19	259.97	64.93	1543.12	80.41	1241.60	40.63
Claude-Opus-4.8	198.45	8.44	147.97	240.44	61.81	1776.13	76.78	965.12	50.97
Claude-Fable-5	288.14	11.73	418.94	238.61	44.52	2015.56	77.16	1027.30	131.63
Avengers-Pro ($\alpha = 0.0$)	26.65	**0.55**	**23.81**	14.83	2.23	94.23	2.97	476.23	8.77
Avengers-Pro ($\alpha = 0.53$)	80.11	1.76	250.03	157.28	13.96	454.31	13.07	160.91	14.75
Avengers-Pro ($\alpha = 0.80$)	80.82	1.86	257.34	165.42	14.36	468.45	13.16	176.34	16.31
Router-R2	**10.37**	1.54	129.73	14.12	2.87	98.70	3.36	21.34	14.33
ICL-Router	45.08	2.13	504.72	21.54	13.27	546.42	22.70	102.53	37.87
Avengers	25.87	2.33	126.34	221.79	19.25	573.54	38.24	178.23	31.03
Lens-Router (Ours)	18.68	1.71	27.50	**13.45**	**1.56**	**85.31**	**2.78**	**50.65**	**4.26**

model capabilities within the evaluated model pool. Compared with the strongest accuracy baseline, Avengers-Pro ($\alpha = 0.8$), which achieves an average accuracy of 68.73%, Lens-Router attains higher accuracy while reducing the average realized cost by 82.8%, with the reduction reaching 91.9% on individual benchmarks. Compared with the lowest-cost baseline, Router-R2, Lens-Router further reduces the average realized cost by 30.5% while 16.69% higher accuracy, with the improvement up to 42.0 percentage points on individual benchmarks.

We further evaluate Lens-Router under a more challenging model-pool setting, where the candidate models are deliberately selected to form three capability clusters with closely matched capabilities and official prices, rather than simply using a broad set of newly released frontier models. As shown in Table 3, the selected models are more difficult to distinguish based on their overall capability or nominal price, providing a stricter setting for evaluating fine-grained routing. Under this setting, Avengers-Pro ($\alpha = 0.80$) remains the strongest baseline, achieving an average accuracy of 75.14%, while Lens-Router still achieves higher accuracy. More importantly, Lens-Router reduces the total realized cost on all 5 benchmarks compared with Avengers-Pro ($\alpha = 0.80$), with reductions ranging up to 68.5%. These results show that Lens-Router maintains its quality advantage while achieving more consistent cost savings even when the candidate models are deliberately restricted to closely matched capability and price clusters.

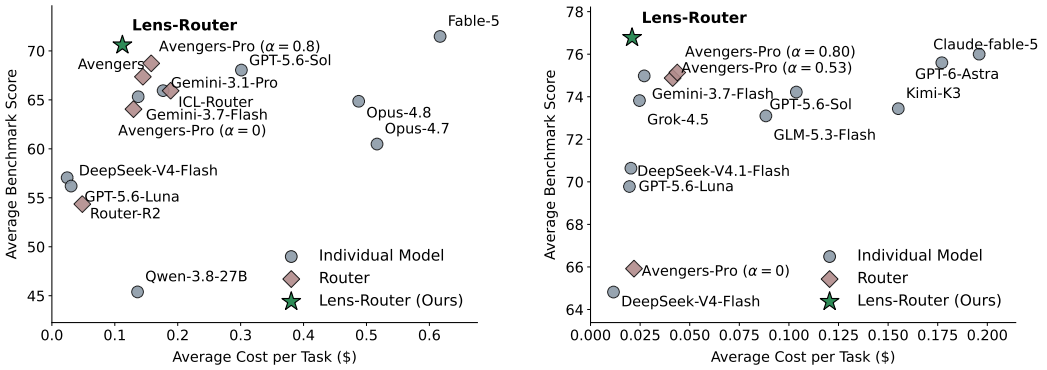
We finally visualize the performance-cost trade-off of individual models and routing methods using the scatter plots in Figure 3, where the x-axis denotes the average cost per task across benchmarks and the y-axis denotes the average benchmark score. In the frontier-model setting, existing routers can generally reach a favorable region of the performance-cost space and approach the Pareto frontier of individual models. However, when the candidate pool consists of closely matched models, the existing routers no longer reach the Pareto frontier. In contrast, Lens-Router consistently achieves a more favorable position in both settings, outperforming existing routers while remaining competitive with or superior to individual models.

Ablation Study We first evaluate whether learning resource demand representations can improve routing performance. We compare ModernBERT-Base, MiniLM-L12-H384, Qwen3-Embedding-0.6B, Qwen3-7B, and XLM-RoBERTa-XL, covering both encoder- and decoder-based backbones with different parameter scales. All models are trained with the same resource demand prediction objective used in our method, while both fine-tuned and pretrained versions are evaluated; for ModernBERT, we additionally test Stage 1 only without the hard-pair shaping in Stage 2.

As shown in Table 4, fine-tuning generally improves routing performance across backbones. The results also show that encoder and decoder architectures achieve comparable performance at moderate parameter scales, while an excessively small backbone may fail to learn resource demand effectively.

Table 3: Accuracy (%) and total realized cost of individual models and LLM routers across different benchmarks. Models with the same background color belong to the same capability cluster.

Model	GPQA-Diamond		HLE		MMLU-Pro		LBM		SciCode	
	Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost
GLM-5.3-Flash	91.7	1.09	46.9	46.73	79.8	11.98	87.9	111.36	59.2	4.03
DeepSeek-V4.1-Flash	92.1	1.78	39.2	90.03	77.4	14.68	88.3	13.85	51.9	3.56
GPT-5.6-Luna	91.1	1.96	33.4	22.52	79.1	12.10	88.0	26.12	51.6	2.67
DeepSeek-V4-Flash	86.8	1.47	34.8	20.44	75.5	12.27	86.8	12.14	40.2	1.75
Gemini-3.7-Flash	94.5	1.97	47.9	79.36	85.6	13.11	87.1	29.71	59.8	2.89
Grok-4.5	94.2	3.25	42.7	142.70	80.9	29.02	90.8	6.62	56.5	7.48
GPT-5.6-Sol	94.1	14.47	42.2	579.17	81.1	21.39	96.3	51.15	57.4	22.90
Kimi-K3	93.7	23.94	48.2	849.40	81.4	150.13	84.4	91.63	59.5	5.35
Avengers-Pro ($\alpha = 0.0$)	86.9	1.89	37.1	**20.23**	75.7	**12.14**	87.1	29.96	42.8	**2.18**
Avengers-Pro ($\alpha = 0.53$)	93.2	2.78	46.5	80.67	80.3	19.32	95.6	48.42	58.8	7.65
Avengers-Pro ($\alpha = 0.80$)	94.5	1.97	47.2	81.72	80.3	27.89	**95.6**	48.42	58.1	11.87
Router-R2	91.8	**1.59**	46.2	51.53	79.2	18.11	87.9	101.64	59.1	4.17
ICL-Router	93.9	13.84	42.3	560.33	80.6	22.08	95.3	97.10	57.4	22.83
Lens-Router (Ours)	**95.1**	1.73	**48.1**	52.03	**86.8**	14.23	94.7	**21.74**	**59.2**	3.74



(a) Frontier models with capability-based grouping.

(b) Closely matched models.

Figure 3: Performance–cost distributions under different model-pool compositions. (a) Frontier models are selected and grouped according to their capability levels. (b) The model pool focuses on closely matched models whose capabilities are difficult to distinguish.

Conversely, increasing the backbone size to 7B or 3.5B provides no meaningful advantage over the much smaller ModernBERT, suggesting that substantially larger models are unnecessary for learning the resource demand representation. Finally, without fine-tuning, the semantic representations of different pretrained backbones yield relatively similar performance, indicating that the observed gains primarily come from resource demand-oriented supervision rather than the choice of semantic encoder alone.

To examine the effectiveness of the two-stage routing design, we compare Lens-Router with several alternative decision architectures under varying amounts of historical supervision while keeping the resource demand encoder fixed. Specifically, Cluster $\rightarrow$ Pairwise denotes our complete design, which first filters capability-compatible model clusters and then performs query-conditioned pairwise cost ranking within the remaining candidates. Individual $\rightarrow$ Pairwise removes capability clustering and directly predicts the capability of each individual model; the five highest-ranked models are then passed to Stage 3 for pairwise cost ranking. Cluster $\rightarrow$ Direct keeps the cluster-based capability filtering but replaces pairwise cost ranking with direct relative-ranking prediction. Joint MLP jointly models capability and cost with a single MLP and directly outputs a model ranking, while Joint Clustering uses a unified clustering procedure without separating the capability and cost stages.

Table 4: Ablation study of the resource demand encoder.

Encoder	Training	Acc. ↑	Cost/Task ↓
ModernBERT (149M)	Full	70.8	0.13
	Stage 1 only	70.3	0.15
	None	66.4	0.19
Qwen3-Embedding-0.6B (0.6B)	Full	70.7	0.13
	None	68.3	0.20
Qwen3-7B (7B)	Full	70.9	0.15
	None	68.6	0.35
XLM-RoBERTa-XL (3.5B)	Full	70.6	0.14
	None	64.3	0.19
MiniLM-L12-H384 (33M)	Full	62.9	0.37
	None	64.6	0.26

Table 5: Ablation study of the routing design under different amounts of historical supervision.

Variant	Routing Design	300 Queries		500 Queries		700 Queries		900 Queries	
		Acc. ↑	Cost ↓	Acc. ↑	Cost ↓	Acc. ↑	Cost ↓	Acc. ↑	Cost ↓
Ours	Clu → Pair	65.7	0.79	68.5	0.12	70.8	0.13	70.9	0.13
w/o Clustering	Indi → Pair	66.9	1.33	67.3	0.70	70.7	0.25	70.8	0.25
w/o Pairwise	Clu → Direct	65.6	0.83	68.5	0.39	70.8	0.21	70.6	0.20
Joint MLP	Joint MLP	66.7	1.50	68.1	0.27	69.9	0.15	70.0	0.16
Joint Clustering	Joint Clustering	68.1	0.86	69.5	0.23	69.8	0.15	70.1	0.15

As shown in Table 5, with limited historical supervision, all routing variants exhibit degraded performance, while increasing the supervision size generally improves their accuracy and cost efficiency. The jointly modeled variants are particularly sensitive to supervision size, showing more pronounced improvements as additional historical data are provided. With sufficient supervision, removing capability clustering or pairwise ranking preserves similar accuracy but substantially increases cost, from 0.13 to 0.25 and 0.21, respectively. In contrast, jointly modeling capability and cost remains less effective, with lower accuracy than the complete design, demonstrating the benefit of explicitly decomposing routing into capability filtering and subsequent cost ranking.

Combining the two ablation studies shows that the main performance gain comes from learning the resource demand representation rather than relying on pretrained semantic representations alone, which helps alleviate the noise and ambiguity inherent in natural-language queries. The subsequent two-stage routing design further improves the quality-cost trade-off by first identifying capability-compatible candidates and then performing fine-grained cost comparison among them.

5 Conclusion and Discussion

In this work, we propose Lens-Router, a progressive fine-grained router that learns a reusable query-level resource demand representation and progressively refines model selection through capability filtering and cost ranking. Our formulation does not assume that resource demand is an absolute or fully model-independent property, but rather exploits the relative demand structure that remains sufficiently stable across models. The scalability of Lens-Router comes from reusing this learned query representation when new models are introduced, so that only limited model-specific supervision is required instead of relearning the routing representation from scratch. Throughout our experiments, we fix the inference configuration by setting temperature to 0 and using the default reasoning effort for all models, leaving the effects of varying inference settings for future investigation. Overall, Lens-Router provides a scalable and fine-grained approach to model selection in an increasingly diverse LLM ecosystem.

AI USAGE STATEMENT

In this work, we used generative AI tools to construct supervision data for the resource demand dataset and to improve the language and readability of the manuscript. LLMs were used to score resource demand dimensions based on queries, model execution results, and predefined criteria. All AI-assisted work was reviewed by the authors, who take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Theoretical analyses and proofs are provided in Appendix A.3, empirical analyses of resource-demand stability and capability boundaries in Appendices A.2 and A.1, and details of resource-demand dimension selection, dataset construction, evaluation prompts, validation, and encoder training in Appendices B.1.1, C, B.1.2, and B.1.3. The complete routing procedure, model clusters, and pair configurations are provided in Appendix B.2.2. The prompt used for generating resource demand dimensions is given in Appendix C. Upon acceptance, we will release the source code, model checkpoints, and datasets to facilitate reproduction of our results.

REFERENCES

Shubham Agrawal and Prasang Gupta. Llmrank: Understanding llm strengths for model routing. *arXiv preprint arXiv:2510.01234*, 2025.

Keivan Alizadeh-Vahid, Seyed Iman Mirzadeh, Hooman Shahrkokhi, Dmitry Belenko, Frank Sun, Minsik Cho, Mohammad Hossein Sekhavat, Moin Nabi, and Mehrdad Farajtabar. Duo-llm: A framework for studying adaptive computation in large language models. In *NeurIPS Efficient Natural Language and Speech Processing Workshop*, pp. 443–455. PMLR, 2024.

Dhananjay Ashok and Jonathan May. Language models can predict their own behavior. volume 38, pp. 62708–62739, 2026.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Advances in Neural Information Processing Systems*, 38, 2026.

Robert E Blackwell, Jon Barry, and Anthony G Cohn. Towards reproducible llm evaluation: quantifying uncertainty in llm benchmark scores (2024). *URL https://arxiv. org/abs/2410.03492*, 2024.

Peizhuang Cong, Tong Yang, Yuchao Zhang, Wendong Wang, and Ke Xu. Mico: efficient query scheduling for multi-cloud deployed llm inference service. *Science China Information Sciences*, 69(3):132102, 2026.

Mert Demirer, Andrey Fradkin, and Nadav Tadelis. The emerging market for intelligence: How firms buy and sell ai. *Journal of Economic Perspectives*, 40(3):23–46, 2026.

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.

Qi Fan, An Zou, and Yehan Ma. Timebill: Time-budgeted inference for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30620–30628, 2026.

Yujia Fu, Heming Zhong, Dan Huang, and Yutong Lu. Flare: Fine-grained length-aware routing for resource-efficient heterogeneous llm serving. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22249–22266, 2026.

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. *arXiv preprint arXiv:2403.12031*, 2024.

Declan Jackson, William Keating, George Cameron, and Micah Hill-Smith. Aa-omniscience: Evaluating cross-domain knowledge reliability in large language models. *arXiv preprint arXiv:2511.13029*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

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.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, et al. Llmrouterbench: A massive benchmark and unified framework for llm routing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37733–37754, 2026.

Junnan Liu, Hongwei Liu, Linchen Xiao, Ziyi Wang, Kuikun Liu, Songyang Gao, Wenwei Zhang, Songyang Zhang, and Kai Chen. Are your llms capable of stable reasoning? In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 17594–17632, 2025.

Xue Liu, Xin Ma, Yuxin Ma, Yongchang Peng, Duo Wang, Zhoufutu Wen, Ge Zhang, Kaiyuan Zhang, Xinyu Chen, Yida Ding, et al. Xpertbench: Expert level tasks with rubrics-based evaluation. *arXiv preprint arXiv:2604.02368*, 2026a.

Xunzhuo Liu, Huamin Chen, Samzong Lu, Yossi Ovadia, Guohong Wen, Hao Wu, Zhengda Tan, Jintao Zhang, Senan Zedan, Yehudit Kerido, et al. vllm semantic router: Signal driven decision routing for mixture-of-modality models. *arXiv preprint arXiv:2603.04444*, 2026b.

Kenichirou Narita, Siqi Peng, Taku Fukui, Moyuru Yamada, Satoshi Munakata, and Satoru Takahashi. Overcoming the" impracticality" of rag: Proposing a real-world benchmark and multi-dimensional diagnostic framework. *arXiv preprint arXiv:2604.02640*, 2026.

Bitya Neuhof and Yuval Benjamini. Quantifying ranking uncertainty in llm benchmarks. *arXiv preprint arXiv:2607.16259*, 2026.

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.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Grzegorz Piotrowski, Mateusz Bystroński, Mikołaj Hołysz, Jakub Binkowski, Grzegorz Chodak, and Tomasz Jan Kajdanowicz. When will the tokens end? graph-based forecasting for llms output length. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 843–848, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

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. *arXiv preprint arXiv:2311.12022*, 2023.

Jim Samuel, Meng Ye, Tanya Khanna, Yi Yao, Xiao Lin, Rick Anderson, and Anish Gupta. Finder, evaluator, explainer, generator (feeg): A bloom's taxonomy-based query classification framework for llms and generative ai. *Expert Systems with Applications*, pp. 130755, 2025.

Regolo.ai Team. Brick complexity extractor dataset: Query complexity annotations for llm routing, 2026.

Minyang Tian, Luyu Gao, Shizhuo D Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, et al. Scicode: A research coding benchmark curated by scientists. *Advances in Neural Information Processing Systems*, 37:30624–30650, 2024.

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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33413–33421, 2026a. doi: 10.1609/aaai.v40i39.40628.

Jize Wang, Han Wu, Zhiyuan You, Yiming Song, Yijun Wang, Zifei Shan, Yining Li, Songyang Zhang, Xinyi Le, Cailian Chen, Xinping Guan, and Dacheng Tao. Routemoa: Dynamic routing without pre-inference boosts efficient mixture-of-agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12190–12208. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.558.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Anand Wankhade and Aniket K Shahade. Cost-aware structured content generation using hybrid retrieval-augmented generation and adaptive compute routing. *International Journal of Networked and Distributed Computing*, 2026.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, 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)*, pp. 2526–2547, 2025.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, volume 2025, pp. 91595–91631, 2025.

Yifei Xia, Fangcheng Fu, Wentao Zhang, Jiawei Jiang, and Bin Cui. Efficient multi-task llm quantization and serving for multiple lora adapters. volume 37, pp. 63686–63714, 2024.

Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-router: A new paradigm for llm routing with reasoning. *arXiv preprint arXiv:2602.02823*, 2026.

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, 2025a. doi: 10.52202/085713-4719.

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 *Proceedings of the 2025 7th International Conference on Distributed Artificial Intelligence*, pp. 122–129, 2025b.

Yiqun Zhang, Hao Li, Chenxu Wang, Linyao Chen, Qiaosheng Zhang, Peng Ye, Shi Feng, Xinrun Wang, Jia Xu, Lei Bai, et al. The avengers: A routing recipe for collective intelligence in language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34870–34878, 2026.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In *International Conference on Learning Representations*, volume 2024, pp. 34590–34605, 2024.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. volume 36, pp. 65517–65530, 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.

Table 6: Accuracy and variation of the 17 models appearing across all four benchmarks. The variation is reported as $\pm$ values from the benchmark results. Models are ordered by their ProofBench accuracy.

Model	ProofBench	Vibe-Code	MedScribe	Excel Modeling
GPT-6 Astra	99.00 ± 1.00	89.59 ± 2.17	87.91 ± 1.94	71.70 ± 2.48
Claude Opus 5	99.00 ± 0.99	88.40 ± 3.00	90.98 ± 1.92	73.56 ± 2.24
Claude Fable 5	95.00 ± 2.18	90.35 ± 2.10	88.52 ± 1.95	73.67 ± 2.47
Kimi K3	87.00 ± 3.56	84.96 ± 2.69	87.96 ± 1.89	66.40 ± 3.04
GPT-5.6 Sol	83.00 ± 3.76	80.50 ± 3.72	85.23 ± 1.97	72.34 ± 2.37
Claude Sonnet 5	77.00 ± 4.21	81.33 ± 3.05	76.05 ± 3.05	66.32 ± 3.01
GPT-5.6 Terra	74.00 ± 4.41	74.59 ± 4.20	82.87 ± 1.95	66.20 ± 3.11
GPT-5.6 Luna	60.00 ± 4.90	77.06 ± 3.07	84.39 ± 2.58	67.12 ± 2.94
Gemini 3.7 Flash	58.00 ± 4.96	70.39 ± 4.84	83.94 ± 2.00	71.33 ± 2.26
Qwen 3.8 Max	58.00 ± 4.94	64.70 ± 5.36	84.95 ± 2.00	60.07 ± 3.35
DeepSeek V4 Flash 0731	56.00 ± 4.96	74.74 ± 3.44	80.36 ± 1.97	56.98 ± 3.39
Grok 4.6	51.00 ± 5.02	76.24 ± 3.82	86.53 ± 1.96	62.73 ± 3.08
DeepSeek V4 Pro 0813	50.00 ± 5.03	82.30 ± 3.15	80.17 ± 2.00	52.80 ± 3.06
GLM 5.3	49.00 ± 5.02	78.13 ± 3.94	88.81 ± 2.00	56.34 ± 3.32
Gemini 3.8 Flash	48.00 ± 5.02	78.65 ± 3.88	84.50 ± 1.94	72.20 ± 2.42
Muse Spark 1.2	43.00 ± 4.95	79.10 ± 3.31	90.06 ± 1.96	56.98 ± 3.13
Gemini 3.5 Flash	31.00 ± 4.63	48.68 ± 4.73	76.57 ± 1.92	63.55 ± 2.75

A Empirical Foundations

A.1 Evidence for Cluster-Level Capability Supervision

Many existing LLM routers rely on historical query–model performance data to estimate model capabilities and guide subsequent model selection. However, this assumption becomes increasingly fragile as models with similar capabilities become more common and LLM evaluation exhibits inherent instability. Prior studies have shown that LLMs can produce inconsistent reasoning outcomes across repeated trials (Liu et al., 2025), while benchmark scores remain uncertain even under controlled evaluation settings, including temperature zero and fixed random seeds (Blackwell et al., 2024). Recent work further demonstrates substantial uncertainty in the relative ranking of LLMs across benchmark subsets (Neuhof & Benjamini, 2026). These phenomena imply that historical correctness is not a deterministic indicator of model capability: when candidate models are already close in capability, small differences in historical accuracy may reflect evaluation randomness rather than persistent capability gaps. Consequently, precisely distinguishing which individual model is more capable from historical correctness becomes inherently difficult and sensitive to the particular queries and trials used for supervision. Therefore we perform capability identification at the cluster level.

Table 6 presents the performance of the 17 models that are shared across all four benchmarks. We first examine whether adjacent models can be reliably distinguished from their reported accuracy. For each benchmark, we consider the 16 adjacent model pairs in the ProofBench ordering and check whether the uncertainty intervals of the two models overlap. Across the resulting 64 comparisons, 48 pairs (75.0%) exhibit overlapping intervals, indicating that the observed performance gap between many neighboring models is comparable to the variation of benchmark outcomes itself.

Moreover, the relative ordering of models is highly benchmark-dependent: among the 16 adjacent pairs, 47 of the 96 pairwise comparisons across benchmark pairs reverse their relative ordering, affecting 15 of the 16 adjacent model pairs. These observations suggest that when candidate models have similar overall capabilities, historical correctness data provide an intrinsically noisy and query-dependent basis for fine-grained model discrimination.

This motivates our cluster-level capability selection, which avoids forcing the router to make unstable distinctions among near-equivalent models and instead leaves finer-grained selection to the subsequent cost-aware stage.

A.2 Resource Scaling and Cost Ordering Across Queries and Models

Motivation and Research Questions. Our cost-aware ranking relies on query-conditioned resource consumption, but accurately predicting the absolute number of tokens generated by an LLM is inherently difficult. In autoregressive generation, output length is an outcome of the entire generation trajectory rather than a deterministic function of the input; this difficulty is further amplified by built-in reasoning mechanisms, where both reasoning and final-output lengths can vary substantially across generations. Thus, although token consumption may exhibit stochastic and long-tailed behavior, such variability does not necessarily imply the absence of systematic differences across queries. We choose to measure the reasoning and output tokens generated by multiple LLMs for the same set of queries and examine whether their resource consumption exhibits stable, query-dependent structure. Specifically, we investigate two questions: **RQ1: Are resource demands distinguishable across queries despite generation variability?** and **RQ2: Do models exhibit different resource-scaling patterns and query-dependent cost rankings?**

RQ1: Are resource demands distinguishable across queries despite generation variability? We select nine LLMs from different providers and construct two sets of three queries with different resource demands. For each set, we randomly sample one question at each of three difficulty levels from difficulty-labeled mathematical problems and the LeetCode problem set, yielding queries from the same and different domains, respectively. Each model is evaluated on each query for 60 repeated trials, recording the generated output and reasoning tokens at every trial. We then examine the resulting token-consumption distributions and decompose their variance into within-query stochastic variation and between-query systematic variation.

† Resource consumption varies systematically with query difficulty. Although repeated generations exhibit substantial variation, the distributions remain clearly separated across queries with different difficulty levels. As shown in Figures 4 and 5, this separation consistently appears across models, indicating that query difficulty induces systematic differences in resource consumption rather than merely reflecting random generation noise.

† Query-level variation dominates generation-level stochasticity. The systematic differences across queries are substantially larger than the stochastic variation within the same query. As shown in Figure 6, the ratio between between-query and within-query variance ranges from $15.5\times$ to $173.2\times$ across the nine representative models, with all models exhibiting considerably larger between-query variation. This result demonstrates that, despite the stochastic and potentially long-tailed nature of token generation, query-dependent resource demand remains a strong and observable signal.

RQ2: Do models exhibit different resource-scaling patterns and query-dependent cost rankings? Experimental Setup. We construct three model clusters based on their official output-token prices. Cluster 1 contains Qwen-3.7-Max, Grok-4.5, Qwen-3.8-Max, and Gemini 3.7 Flash, with output prices ranging from \$3.75 to \$7.50 per million tokens; Cluster 2 contains Claude Opus 4.6, Claude Opus 4.8, Claude Opus 4.7, GPT-5.5, and GPT-5.6-Sol, ranging from \$20 to \$30; and Cluster 3 contains Kimi-K3, Gemini 3.1 Pro, and GPT-5.6-Terra, ranging from \$12 to \$15. We evaluate these models on 300 queries spanning five domains: Mathematics, Algorithms & Data Structures, Systems & Software Engineering, Programming, and Writing & Communication. For each query–model pair, we record the total generated tokens and generation time to characterize query-dependent resource consumption. As shown in Figure 7, token consumption consistently increases with generation time across models and domains, providing a basis for examining whether different models exhibit distinct resource-scaling patterns.

To characterize the resource demand of each query independently of any particular model, we derive a model-normalized resource demand score from the 300-query evaluation set. For each model m, we first record the total number of tokens generated for all 300 queries and normalize each query by the maximum token consumption observed for that model. Specifically, if $T_{q,m}$ denotes the total generated tokens for query q by model m, we define

$$r_{q,m} = \frac{T_{q,m}}{\max_{q'} T_{q',m}}. \tag{5}$$

Thus, the query that induces the largest resource consumption for model m is assigned a value of 1, while all other queries receive a relative resource-consumption ratio. Each query therefore obtains

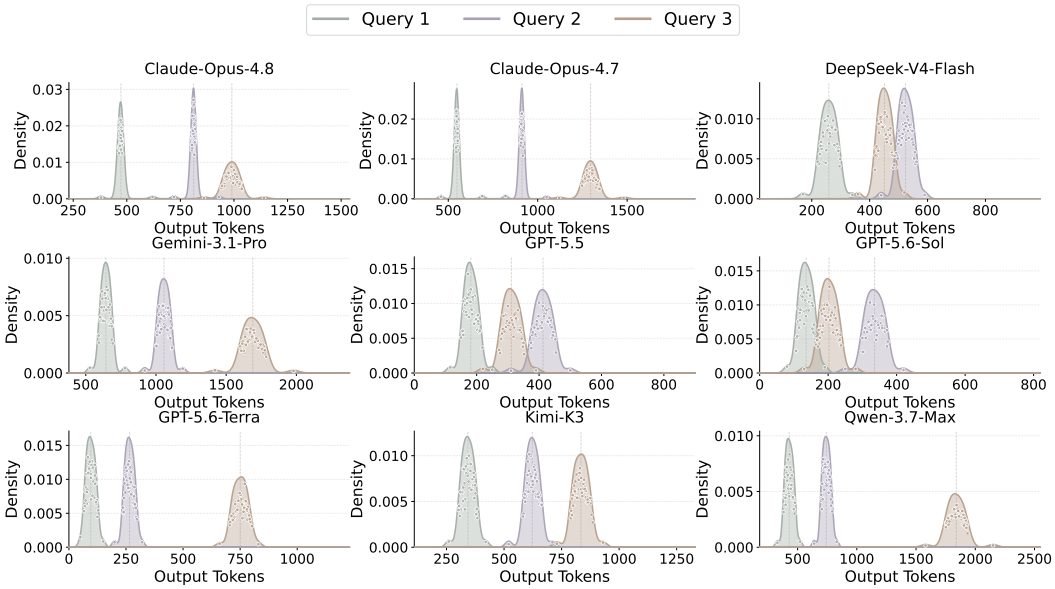


Figure 4: Output-token distributions of nine representative LLMs for three queries with different resource demands within the same domain. Each point represents an individual generation, and the density curves summarize the corresponding output-token distributions. Despite sharing the same domain, the three queries exhibit clearly separated resource consumption patterns across models, indicating that resource demand is not determined solely by the domain but varies substantially across individual queries.

a set of normalized resource demand scores across the evaluated models. We then average these scores across models:

$$R_q = \frac{1}{M} \sum_{m=1}^{M} r_{q,m}, \tag{6}$$

where M is the number of evaluated models. R_q serves as the query-level resource demand measure used on the horizontal axis of Figures 8, 9 and 10. By normalizing within each model before aggregation, this measure reduces the influence of model-specific generation-length scales and captures the relative resource demand induced by the query itself.

† Cross-model resource demand is highly consistent across queries. Despite substantial differences in model capabilities and generation behavior, a query that induces relatively high resource consumption on one model tends to induce similarly high consumption on other models. Across the 300-query evaluation set, 98.3% of queries differ by no more than 0.05 in their normalized resource demand scores across models. This consistency indicates that the relative resource demand induced by a query is largely stable across models, supporting its use as a transferable query-level signal for downstream routing.

† Resource expansion is systematic but model- and domain-dependent. Across the examined models, resource consumption generally increases with query resource demand, confirming the existence of systematic resource expansion. However, the scaling rate varies substantially across models and domains. For example, Qwen-3.7-Max and Grok-4.5 exhibit slopes of 297.62 and 84.96 in mathematics, but 68.44 and 164.51 in writing, respectively. This indicates that resource expansion is not governed by a model-specific fixed rule, but depends on the interaction between the model and the query.

† Cost-order reversals are widespread across models, providers, and price tiers. Resource-scaling heterogeneity frequently causes the relative cost ordering between models to reverse as query demand increases. Among the 95 model-pair–domain comparisons, 30 exhibit scaling-curve crossings, corresponding to 31.6% of all comparisons. The phenomenon appears across all three price-aligned clusters, with crossing rates of 50.0%, 26.0%, and 13.3% in Clusters 1–3, respectively. These re-

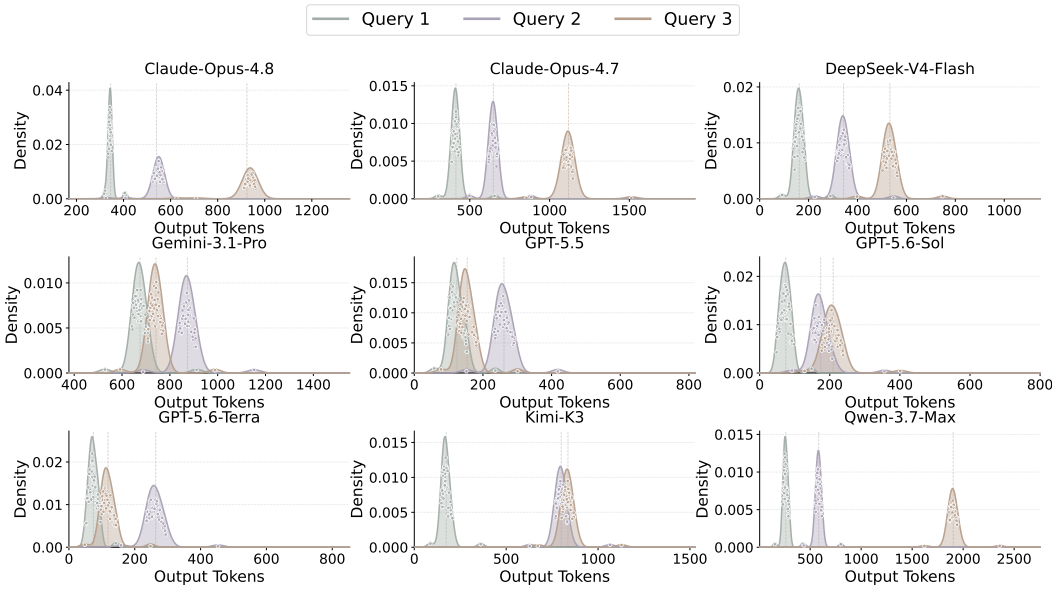


Figure 5: Output-token distributions of nine representative LLMs for three queries with different resource demands across different domains. Each point represents an individual generation, and the density curves summarize the corresponding output-token distributions. The separation of the distributions across queries persists under domain variation, suggesting that query-dependent resource demand remains observable beyond domain-specific effects.

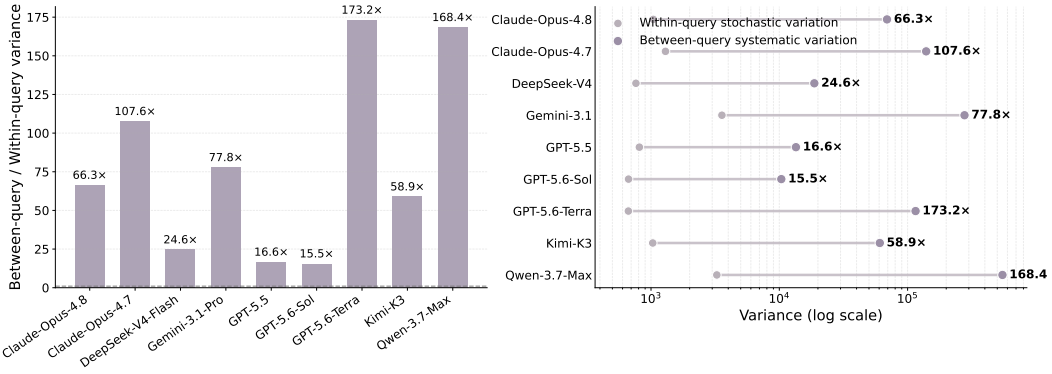


Figure 6: Variance decomposition of output-token consumption across representative LLMs. The left panel reports the ratio between between-query systematic variation and within-query stochastic variation. The right panel presents the two sources of variation on a logarithmic scale. Across all models, between-query variation substantially exceeds within-query variation, demonstrating that resource expansion is dominated by systematic query-dependent behavior rather than random generation noise.

versals occur across models from different providers, demonstrating that they are not specific to a particular vendor or price tier.

† Relative cost ordering is strongly domain-dependent. The resource-efficiency relationship between two models can change substantially across domains. The same model pair may exhibit a reversal in one domain while maintaining a stable ordering in another. For example, Qwen-3.7-Max and Grok-4.5 reverse in mathematics, coding, and writing, exhibit a near-parallel crossing in systems, but do not cross in algorithms. Similarly, GPT-5.6-Terra and Gemini 3.1 Pro show no crossing in mathe-

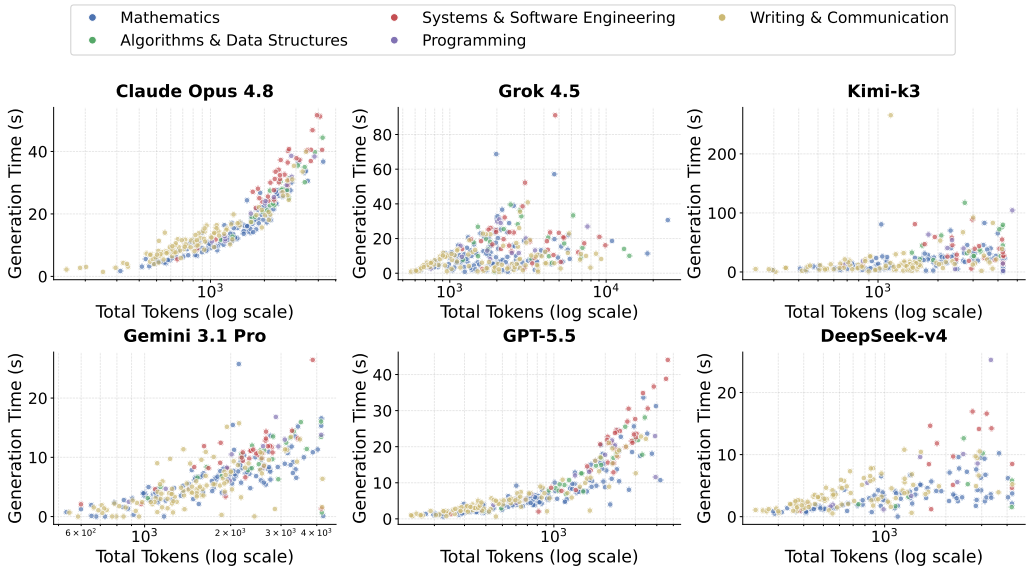


Figure 7: Relationship between token consumption and generation time across 300 queries for six representative LLMs. Each point corresponds to one query and is colored according to its domain. The consistent increasing trends across models show that query-dependent resource expansion is reflected not only in token consumption but also in serving time.

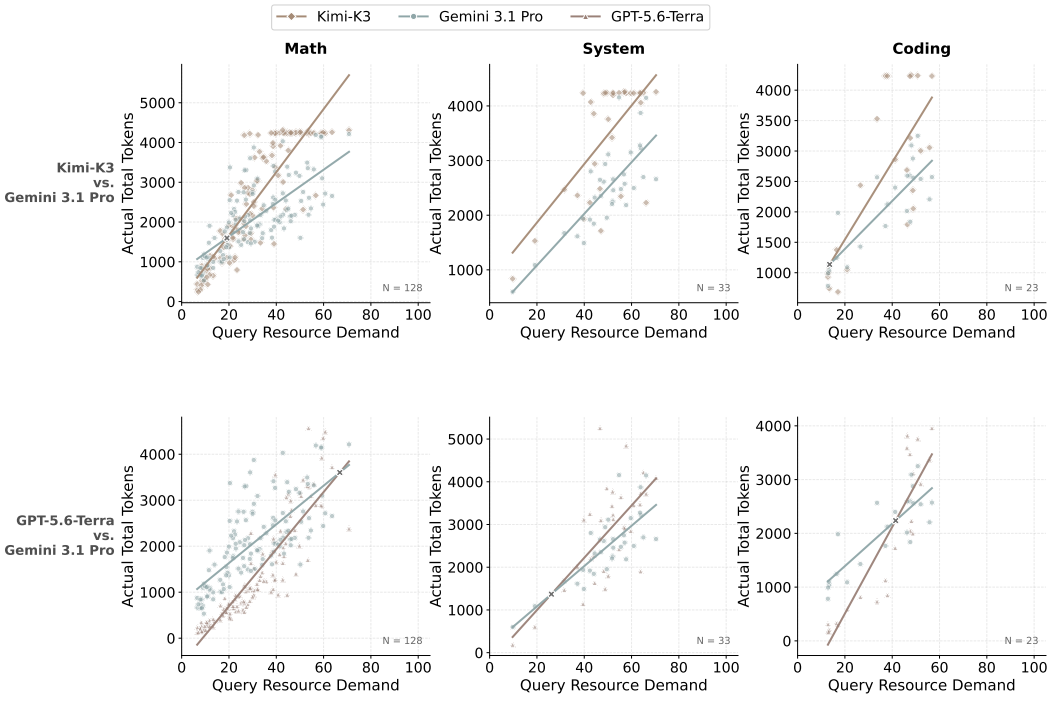


Figure 8: Resource expansion patterns of selected model pairs in Cluster 3. Each row represents a model pair, while the columns correspond to the mathematical, system, and coding domains.

matics, systems, algorithms, or writing, but exhibit a pronounced reversal in coding. These results indicate that model cost ordering cannot be reliably determined from model-level characteristics alone, but must account for query-specific properties.

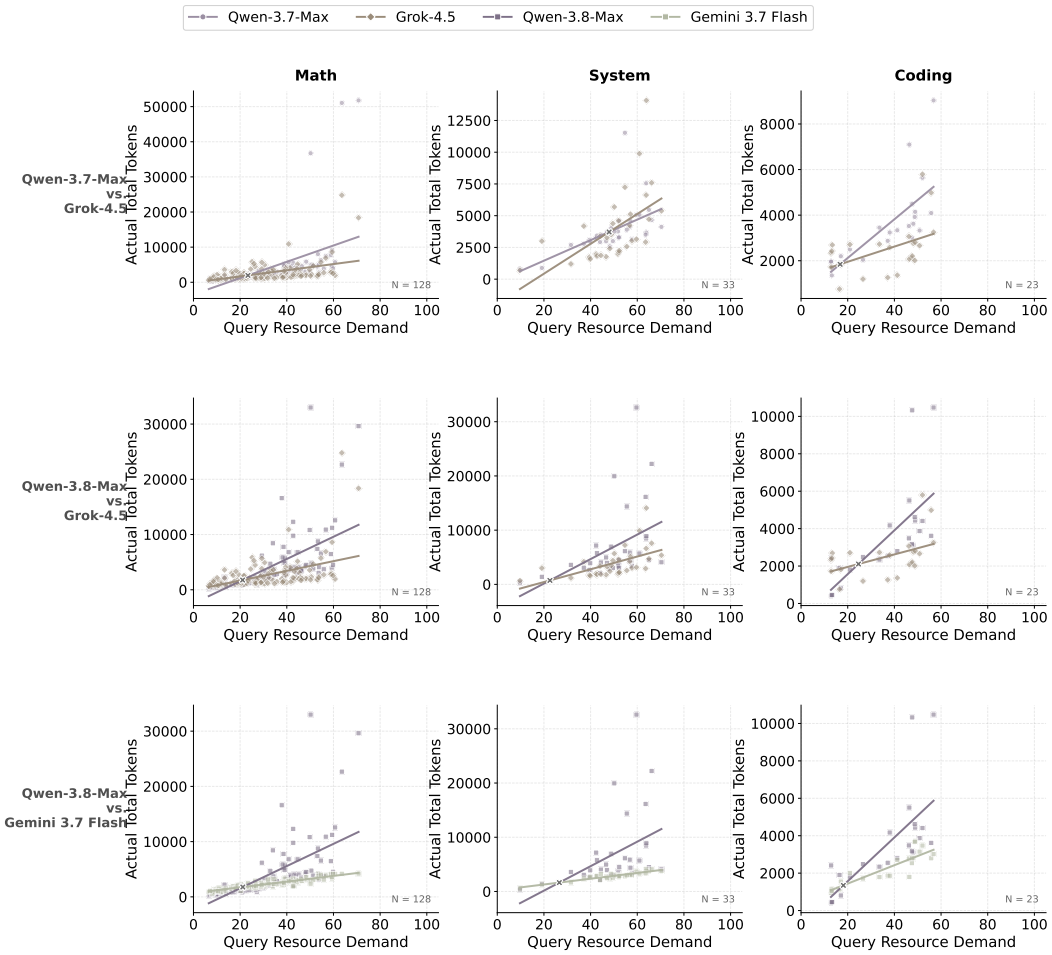


Figure 9: Resource expansion patterns of selected model pairs in Cluster 1. Each row represents a model pair, while the columns correspond to the mathematical, system, and coding domains.

† Ignoring token-length differences can substantially distort cost comparisons. Nominal output-token prices provide only a coarse indication of realized cost, as models can consume substantially different numbers of tokens for the same query. For example, Qwen-3.7-Max and Grok-4.5 have output prices of \$7.50 and \$6.00 per million tokens, respectively, yet their coding-domain scaling slopes are 406.11 and 38.44. Combining price and scaling rate yields an approximately $13.2\times$ difference in implied cost growth per unit of query demand, despite only a $1.25\times$ difference in nominal price. Conversely, in writing, Qwen-3.7-Max is nominally 25% more expensive but has an implied cost growth rate only 52.0% of Grok-4.5's, making it approximately 48% cheaper under the fitted scaling trend. These results show that routing based solely on nominal prices can substantially misrepresent realized cost.

A.3 Formal Analysis of Semantic Insufficiency and Representation Reuse

This section provides a formal analysis of two key properties motivating Lens-Router. First, semantic representations alone may be insufficient for fine-grained routing when semantically similar queries induce different resource demands and consequently different optimal routing decisions. Second, when query behavior across models is governed by a shared latent resource demand state, explicitly learning this state allows the query representation to be reused across models, reducing the amount of model-specific supervision required for downstream routing.

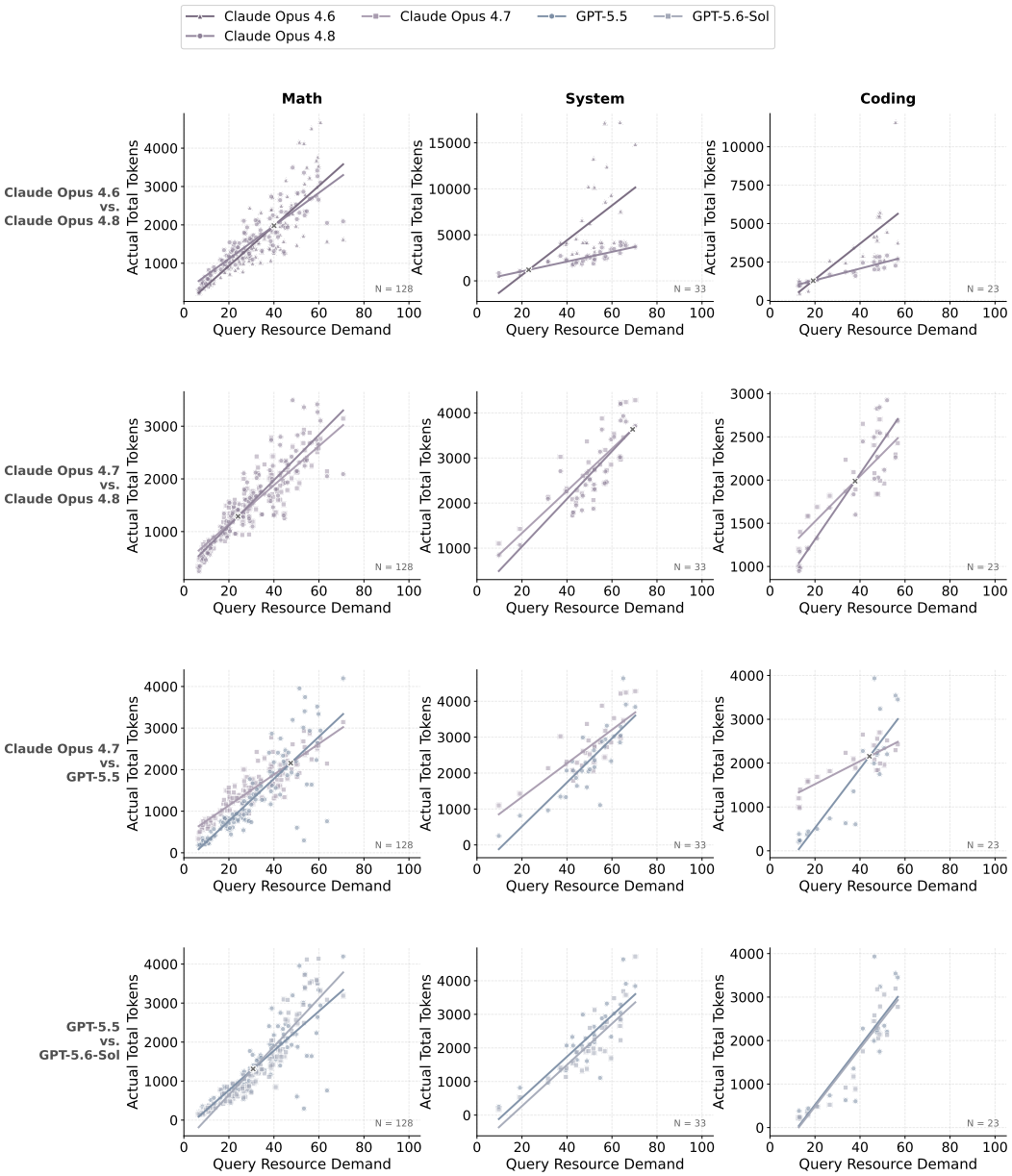


Figure 10: Resource expansion patterns of selected model pairs in Cluster 2. Each row represents a model pair, while the columns correspond to the mathematical, system, and coding domains.

A.3.1 Problem Formulation

Let $q \in \mathcal{Q}$ denote a query, $m \in \mathcal{M}$ a candidate model, and $D(q) \in \mathcal{D}$ the latent resource demand state induced by q. We denote by

$$S(q) = E_{\text{sem}}(q)$$

a generic semantic representation of the query, and by

$$Z(q) = E_{\text{dem}}(q)$$

the representation produced by a resource demand encoder.

For a query–model pair (q, m), let $U(q, m)$ denote its routing utility, which may jointly capture response quality and realized cost. The optimal routing decision is therefore

$$m^*(q) = \arg\max_{m \in \mathcal{M}} U(q, m). \tag{7}$$

We assume that the query-dependent behavior relevant to routing can be described through a shared latent demand state. Specifically, there exists a model-specific function g_m such that

$$U(q, m) = g_m(D(q)). \tag{8}$$

This assumption does not require the resource demand state to be independent of query semantics; rather, it states that $D(q)$ captures the routing-relevant variation that can be shared across different models.

A.3.2 Semantic Representations Are Insufficient for Fine-Grained Routing

A semantic representation may preserve rich information about the query while still failing to preserve the distinctions required for routing. We formalize this observation through semantic collisions.

Definition 1 (Routing-distinguishable queries). Two queries $q_1, q_2 \in \mathcal{Q}$ are *routing-distinguishable* if there exist two candidate models m_a, m_b such that

$$U(q_1, m_a) > U(q_1, m_b) \quad \text{and} \quad U(q_2, m_a) < U(q_2, m_b). \tag{9}$$

Thus, the preferred model reverses between the two queries.

Definition 2 (Semantic collision). For a semantic encoder E_{sem}, two queries q_1, q_2 form an ε-semantic collision if

$$\|S(q_1) - S(q_2)\| \le \varepsilon. \tag{10}$$

The following proposition formalizes the limitation of any router whose decision depends only on semantic representations.

Proposition 1 (Irreducible error under semantic collision). Suppose there exist routing-distinguishable queries q_1, q_2 with

$$S(q_1) = S(q_2).$$

Then any deterministic router of the form

$$f_S : \mathcal{S} \to \mathcal{M}$$

must incur an error on at least one of q_1 and q_2 with respect to the optimal routing decision. If q_1 and q_2 occur with probabilities p and $1 - p$, respectively, then the minimum achievable routing error satisfies

$$\inf_{f_S} \Pr[f_S(S(q)) \neq m^*(q)] \ge \min\{p, 1 - p\}. \tag{11}$$

Proof. Because $S(q_1) = S(q_2)$, any deterministic semantic-only router must assign the same model to both queries:

$$f_S(S(q_1)) = f_S(S(q_2)).$$

However, by equation 9, the optimal models for q_1 and q_2 are different. Hence the router must be incorrect on at least one of the two queries. Assigning the optimal model for q_1 causes an error on q_2 with probability $1 - p$, while assigning the optimal model for q_2 causes an error on q_1 with probability p. The smaller of these two error rates is $\min\{p, 1 - p\}$, which proves equation 11.

Interpretation. Proposition 1 shows that the limitation is not caused by the choice of a particular clustering or prediction algorithm. Whenever distinct routing decisions are mapped to the same semantic representation, no downstream predictor can recover the missing distinction. In practice, exact semantic collisions are replaced by near-collisions, such as queries that are highly similar in language but require substantially different levels of reasoning or generation. Our hard-pair construction in Section 3 is designed to expose precisely such cases.

A.3.3 Explicit Demand Representation

The previous proposition motivates learning a representation that explicitly captures routing-relevant resource demand. We assume that there exist queries q_1, q_2 for which

$$\|S(q_1) - S(q_2)\| \le \varepsilon, \qquad \|D(q_1) - D(q_2)\| \ge \gamma, \tag{12}$$

for some $\gamma > \varepsilon$. This condition captures the common situation in which semantic similarity does not imply similar resource demand, e.g., asking for the meaning of a theorem versus asking for a rigorous proof of the theorem.

A resource demand encoder aims to learn a representation

$$Z(q) \approx D(q),$$

so that routing-relevant distinctions are explicitly represented in the embedding space. The purpose of Stage 1 is to provide absolute demand supervision, while Stage 2 further shapes the representation by enforcing relative demand relations between hard query pairs.

A.3.4 Shared Latent Demand Enables Sample-Efficient Adaptation

We next formalize the scalability benefit of learning a reusable demand representation. Consider K candidate models. A direct query-based router learns K model-specific functions

$$f_m : \mathcal{Q} \to \mathcal{Y}, \qquad m = 1, \dots, K, \tag{13}$$

whereas a demand-aware router first learns a shared representation

$$E_{\mathrm{dem}} : \mathcal{Q} \to \mathcal{D}$$

and subsequently learns

$$g_m : \mathcal{D} \to \mathcal{Y}. \tag{14}$$

The key distinction is that the first mapping is shared across all models, while only the second mapping is model-specific.

Assumption 1 (Shared latent demand). There exists a latent demand variable $D(q)$ such that

$$Y_m \perp q \mid (D(q), m), \tag{15}$$

where Y_m denotes the routing-relevant outcome of model m.

Assumption 2 (Lower complexity in demand space). Let $\mathcal{F}_{\mathcal{Q}}$ denote the hypothesis class for directly mapping queries to routing outcomes and $\mathcal{F}_{\mathcal{D}}$ the hypothesis class for mapping demand representations to routing outcomes. Assume

$$\mathfrak{C}(\mathcal{F}_{\mathcal{D}}) < \mathfrak{C}(\mathcal{F}_{\mathcal{Q}}), \tag{16}$$

where $\mathfrak{C}(\cdot)$ denotes an appropriate statistical complexity measure, such as VC dimension or Rademacher complexity.

Proposition 2 (Sample-efficient model adaptation). Under Assumptions 1–2, for a standard supervised learner with n training samples, the generalization error of a model-specific predictor in demand space admits a bound of the form

$$\mathcal{E}_{\mathcal{D}} \le \widehat{\mathcal{E}}_{\mathcal{D}} + O\left(\sqrt{\frac{\mathfrak{C}(\mathcal{F}_{\mathcal{D}}) + \log(1/\delta)}{n}}\right), \tag{17}$$

with probability at least $1 - \delta$. In contrast, direct query-based learning satisfies

$$\mathcal{E}_{\mathcal{Q}} \le \widehat{\mathcal{E}}_{\mathcal{Q}} + O\left(\sqrt{\frac{\mathfrak{C}(\mathcal{F}_{\mathcal{Q}}) + \log(1/\delta)}{n}}\right). \tag{18}$$

Therefore, for comparable empirical error, the demand-space predictor requires no more samples asymptotically and can require strictly fewer samples whenever

$$\mathfrak{C}(\mathcal{F}_{\mathcal{D}}) < \mathfrak{C}(\mathcal{F}_{\mathcal{Q}}).$$

Proof. The bounds in equation 17 and equation 18 follow from standard uniform generalization bounds for hypothesis classes of finite statistical complexity. Under Assumption 2, the estimation term of the demand-space predictor is strictly smaller for the same sample size. Equivalently, for a target generalization error ϵ, the sufficient sample size scales as

$$n_{\mathcal{D}} = O\left(\frac{\mathfrak{C}(\mathcal{F}_{\mathcal{D}})}{\epsilon^2}\right)$$

and

$$n_{\mathcal{Q}} = O\left(\frac{\mathfrak{C}(\mathcal{F}_{\mathcal{Q}})}{\epsilon^2}\right).$$

Hence

$$n_{\mathcal{D}} < n_{\mathcal{Q}}$$

whenever

$$\mathfrak{C}(\mathcal{F}_{\mathcal{D}}) < \mathfrak{C}(\mathcal{F}_{\mathcal{Q}}).$$

A.3.5 Implications for Model-Pool Scalability

Proposition 2 also characterizes the benefit of adding new models. Suppose the resource demand encoder has already been trained and fixed. For a newly introduced model m_{K+1}, the demand-aware router only needs to learn

$$g_{K+1} : \mathcal{D} \to \mathcal{Y}, \tag{19}$$

rather than relearning a new query-to-model mapping

$$f_{K+1} : \mathcal{Q} \to \mathcal{Y}.$$

Thus, the representation learning cost is amortized over the model pool:

$$\underbrace{\mathcal{Q} \to \mathcal{D}}_{\text{shared and learned once}} \quad + \quad \underbrace{K \text{ model-specific predictors}}_{\text{lightweight adaptation}}. \tag{20}$$

This decomposition provides a formal explanation for why the resource demand encoder can be reused when the model pool evolves. The encoder need not be retrained whenever a new model is added; only the model-specific mapping from the shared demand space to the new model's behavior must be estimated.

Remark. The above result does not require the semantic representation to contain no resource demand information. A pretrained semantic encoder may already encode a weak or entangled signal related to $D(q)$. The advantage of explicit demand supervision is instead that the routing-relevant component is concentrated into a representation whose downstream hypothesis class can be substantially simpler. Consequently, the proposed encoder can improve sample efficiency even when the underlying resource demand information is already partially present in a pretrained representation.

A.3.6 Connection to Lens-Router

The formal analysis above motivates the hierarchical structure of Lens-Router. The resource demand encoder learns a reusable representation

$$q \to Z(q) \approx D(q),$$

after which routing is decomposed into two decisions:

$$Z(q) \to \text{capability-compatible candidates} \to \text{relative cost ranking}. \tag{21}$$

The first decision reduces ambiguity among models with substantially different capability levels, while the second increases decision resolution within closely matched candidate groups. Importantly, both decisions operate on the same query-conditioned demand representation rather than requiring a new query–model representation to be learned for each newly introduced model.

Together, Propositions 1–2 provide two complementary explanations for the design of Lens-Router: semantic representations can be insufficient for fine-grained routing when semantically close queries induce distinct routing decisions, while a reusable latent demand representation can reduce the statistical burden of model-specific adaptation as the model pool grows.

B Implementation Details of Lens-Router

B.1 Details of Resource Demand encoder

B.1.1 Details of Dimension Selection

Prior studies have shown that queries contain information that can be used to characterize their underlying task and resource requirements (Agrawal & Gupta, 2025; Liu et al., 2026b; Alizadeh-Vahid et al., 2024). For example, existing work has explored predicting response length from input queries, demonstrating that query-side information can provide useful signals for anticipating generation-related resource requirements. Other studies have investigated query difficulty and task complexity as signals for adaptive model selection, showing that queries can be characterized according to the amount and type of capability required to solve them. These studies indicate that relevant properties of a query can be extracted before model execution, motivating us to construct a broader set of query-level dimensions for characterizing the resources required to fulfill a query.

We construct the initial candidate set by combining literature investigation with LLM-assisted query analysis. We first review prior studies on query complexity, task difficulty, reasoning requirements, generation requirements, and related properties, and collect dimensions that can potentially reflect the requirements induced by a query. We further use LLMs to analyze representative queries with substantial variation in their answering requirements, identifying additional properties that are not explicitly covered by the dimensions collected from the literature. We consolidate and refine the resulting candidates to remove obvious overlaps and unify dimensions describing similar requirements. This process produces an initial set of 67 candidate resource demand dimensions. The complete set of candidate dimensions, together with their sources and descriptions, is presented in Table 11.

We then empirically screen the 67 candidate dimensions using 3,000 queries that are independent of the data used to train the resource demand encoder. The queries are collected from diverse sources, including MATH, MMLU, GPQA, and WildChat, to cover a broad range of tasks and answering requirements. We first execute each query on three LLMs, GPT-5.5, Claude Opus 4.8, and DeepSeek-V4-Flash, and record the generated responses together with their corresponding resource consumption. These execution results provide concrete evidence about how different queries are handled and the resources required during generation. We subsequently provide the queries, their observed execution results, and the 67 candidate dimensions to three independent LLM judges, Claude Fable 5, GPT-5.6-Sol, and Gemini 3.1 Pro. Each judge evaluates the extent to which every candidate dimension is required by a query based on the observed execution evidence. We aggregate the resulting scores to obtain a query-by-dimension matrix, where each query is represented by its estimated profile across the 67 candidate resource demand dimensions.

Finally, we perform pairwise correlation analysis over the 67 candidate dimensions using the scores obtained from the 3,000 queries. For each pair of dimensions whose correlation exceeds 0.8, we regard the two dimensions as substantially redundant and retain the one that has lower overall correlation with the remaining dimensions. We repeat this procedure until no pair of retained dimensions exceeds the correlation threshold. This process reduces the initial 67 candidates to 24 dimensions, yielding a compact set of relatively independent dimensions that broadly characterize query resource demand. The final 24 dimensions are highlighted in bold in Table 11.

B.1.2 Validation of resource demand Representation

To examine whether the proposed resource demand representation provides a reliable and behaviorally meaningful description of query requirements, we conduct three complementary experiments.

Inter-Annotator Agreement We first investigate whether the proposed 24-dimensional resource demand schema can be interpreted consistently by different LLM annotators. Since the supervision used to train the resource demand Encoder is derived from LLM-based annotations, substantial disagreement among annotators would undermine the reliability of the resulting representation.

We use the same query corpus employed to construct the resource demand supervision and obtain independent annotations from three frontier LLMs: DeepSeek-V4-Pro, GPT-5.6-Sol, and Claude

Table 7: Inter-annotator agreement among the three LLM-based resource demand annotators. We report Spearman correlation (ρ), MAE, and RMSE for each resource demand dimension.

Dimension	A–B			B–C			A–C		
	ρ	MAE	RMSE	ρ	MAE	RMSE	ρ	MAE	RMSE
Math Reasoning	0.93	0.38	0.57	0.91	0.42	0.61	0.92	0.40	0.59
Logical Reasoning	0.89	0.47	0.68	0.87	0.51	0.72	0.90	0.45	0.66
Coding	0.94	0.35	0.53	0.92	0.39	0.58	0.93	0.37	0.55
Knowledge	0.86	0.52	0.74	0.84	0.56	0.79	0.88	0.49	0.71
Writing	0.85	0.54	0.77	0.83	0.58	0.81	0.87	0.51	0.74
Translation	0.88	0.46	0.67	0.86	0.49	0.71	0.89	0.44	0.65
Planning	0.91	0.41	0.60	0.89	0.45	0.65	0.90	0.43	0.62
Creative Thinking	0.82	0.61	0.84	0.80	0.65	0.89	0.84	0.59	0.82
Reasoning Depth	0.94	0.34	0.51	0.92	0.38	0.56	0.99	0.36	0.54
Reasoning Steps	0.92	0.39	0.58	0.90	0.43	0.63	0.91	0.41	0.60
Verification Difficulty	0.90	0.43	0.64	0.88	0.47	0.69	0.91	0.41	0.62
Dependency Length	0.87	0.49	0.71	0.85	0.53	0.76	0.88	0.47	0.69
Search Space	0.89	0.45	0.66	0.87	0.49	0.71	0.90	0.43	0.64
Instruction Complexity	0.91	0.40	0.59	0.89	0.44	0.64	0.90	0.42	0.61
Constraint Count	0.93	0.36	0.50	0.91	0.40	0.59	0.92	0.38	0.56
Constraint Dependency	0.90	0.43	0.64	0.88	0.47	0.69	0.89	0.45	0.66
Context Dependency	0.86	0.51	0.73	0.84	0.59	0.78	0.87	0.49	0.71
Ambiguity	0.83	0.58	0.81	0.81	0.62	0.86	0.85	0.56	0.79
Expected Output Length	0.92	0.37	0.56	0.90	0.41	0.61	0.91	0.39	0.58
Expected Explanation Density	0.88	0.46	0.68	0.86	0.50	0.73	0.89	0.44	0.66
Expected Formula Density	0.91	0.40	0.59	0.89	0.44	0.64	0.90	0.42	0.61
Expected Code Density	0.93	0.35	0.52	0.91	0.39	0.57	0.92	0.37	0.54
Expected Structure Complexity	0.89	0.45	0.66	0.87	0.49	0.71	0.90	0.43	0.64
Overall Resource	0.95	0.31	0.47	0.94	0.34	0.51	0.95	0.32	0.49
Average	**0.90**	**0.44**	**0.65**	**0.88**	**0.48**	**0.69**	**0.92**	**0.43**	**0.63**

Annotators: A = DeepSeek-V4-Pro, B = GPT-5.6-Sol, C = Claude Opus 4.8.

Opus 4.8. Each model evaluates every query along the same 24 resource demand dimensions. For each resource demand dimension, we compare the scores produced by every pair of annotators. We report Spearman correlation (ρ) to measure the consistency of their relative judgments, together with mean absolute error (MAE) and root mean squared error (RMSE) to quantify the magnitude of score discrepancies. The comparison is conducted independently for all 24 dimensions and summarized across dimensions.

As shown in Table 7, the three annotators exhibit consistently high agreement across the resource demand dimensions, with only moderate variation among individual dimensions. The agreement is particularly strong for relatively concrete dimensions such as reasoning depth, reasoning steps, coding, and expected output characteristics, while more subjective dimensions exhibit somewhat larger discrepancies. Nevertheless, the overall agreement remains high across all three annotator pairs, suggesting that the proposed resource demand schema captures concepts that can be assessed consistently by different LLMs rather than reflecting the idiosyncratic judgment of a single annotator.

Semantic-Demand Disentanglement We next examine whether resource demand representations capture information that is not adequately reflected by conventional semantic representations. In particular, semantically similar queries may impose substantially different resource requirements, while semantically different queries may nevertheless require comparable computational resources. A useful resource demand representation should distinguish these cases according to their actual resource requirements rather than merely their semantic similarity.

We use the large-scale query corpus collected for our study, together with the corresponding observed model outputs. For each query pair, we obtain both its original ModernBERT semantic representation and its resource demand representation from the trained resource demand Encoder. The observed output lengths of the corresponding model responses are used as an empirical indicator of realized query-level resource requirements. We construct four types of query pairs according

Table 8: Comparison of semantic and resource demand distances under different query-pair conditions. We construct four types of query pairs according to semantic similarity and observed output-length difference. Lower distance indicates greater representation similarity.

Query Pair Type	Semantic Similarity		Output Difference		Representation Distance	
	Threshold	Mean	Threshold	Mean	ModernBERT	resource demand
Sem. Similar + Demand Similar	> 0.95	0.97	$< 5\%$	2.8%	0.03	**0.06**
Sem. Dissimilar + Demand Similar	< 0.50	0.43	$< 5\%$	3.1%	0.57	**0.09**
Sem. Similar + Demand Different	> 0.95	0.96	$> 50\%$	72.4%	0.04	**0.58**
Sem. Dissimilar + Demand Different	< 0.50	0.41	$> 50\%$	75.6%	0.61	**0.62**

Table 9: Behavioral correspondence between resource demand dimensions and realized inference cost. For each model, we compute the Spearman correlation ρ between a query's resource demand score and its realized inference cost. The last column reports the mean correlation across the three models.

resource demand Dimension	DeepSeek-V4-Pro	GPT-5.6-Sol	Claude Opus 4.8	Mean
Reasoning Depth	0.89	0.86	0.93	0.89
Verification Difficulty	0.88	0.82	0.90	0.87
Search Space	0.66	0.71	0.69	0.69
Instruction Complexity	0.75	0.81	0.78	0.78
Constraint Count	0.57	0.63	0.60	0.60
Context Dependency	0.68	0.79	0.75	0.74
Expected Output Length	0.94	0.94	0.95	0.94
Expected Structure Complexity	0.61	0.67	0.64	0.64
Overall Resource	**0.96**	**0.93**	**0.95**	**0.95**
Mean	**0.77**	**0.79**	**0.79**	**0.78**

to two factors: semantic similarity and observed output-length difference. Specifically, we consider (1) semantically similar queries with similar output lengths, (2) semantically dissimilar queries with similar output lengths, (3) semantically similar queries with substantially different output lengths, and (4) semantically dissimilar queries with substantially different output lengths. For each group, we compare the representation distances produced by the original ModernBERT semantic representation and the proposed resource demand representation.

Table 8 shows that the two representations exhibit different sensitivity patterns. When semantically similar queries have substantially different observed output requirements, the original semantic representation tends to remain close, whereas the resource demand representation separates the queries more clearly. Conversely, when semantically different queries exhibit similar observed resource requirements, their resource demand representations can remain relatively close despite the larger semantic distance captured by ModernBERT. These observations indicate that the proposed representation is not simply a reformulation of conventional semantic similarity; instead, it captures query-level demand characteristics that are partially orthogonal to surface-level semantic similarity.

Behavioral Correspondence To assess whether resource demand scores reflect actual inference behavior, we correlate each query's resource demand score with its realized inference cost on multiple models. Using the same real-world query set, we compute the Spearman correlation between each demand dimension and the corresponding cost for each model independently. The results show a strong positive correspondence, with *Overall Resource* achieving a Spearman correlation of up to 0.95. This result provides behavioral evidence that the learned resource demand representation captures actual variation in query-level inference requirements.

B.1.3 Training Process

We fully fine-tune ModernBERT (Warner et al., 2025), a moderately sized pretrained encoder with general-purpose representation capability and useful prior knowledge, to learn the proposed resource demand representation. Our training data are constructed from 59,577 distinct queries collected

from real-user data in WildChat (Zhao et al., 2024) and the Brick Complexity Extractor (Team, 2026). For each query, we first run it on DeepSeek-V4-Pro, GPT-5.6-Sol, and Claude Opus 4.8 to collect their responses and resource consumption. We then provide each LLM with the query, the corresponding execution results, and the predefined evaluation criteria, and ask it to independently score all resource demand dimensions on a 0-10 scale. The judges therefore assess each dimension based on both the query and its observed execution evidence. The arithmetic mean of the three scores for each dimension is used as the final resource demand label. The prompt used for this process is provided in Appendix C.

To further shape the embedding space around resource demand, we construct hard query pairs by combining the original ModernBERT representations, resource demand labels, and observed execution statistics. We organize the hard pairs into two categories. The first category is execution-based hard pairs, which are designed to prevent the encoder from relying on query length as a proxy for resource demand. Specifically, we identify short queries with fewer than 100 input tokens but more than 3,000 generated tokens, and long queries with more than 1,000 input tokens but fewer than 200 generated tokens. These cases provide explicit counterexamples showing that input length alone does not determine the resource requirements of a query. The second category is representation-based hard pairs, which are designed to decouple resource demand from semantic similarity. We construct semantically similar queries whose normalized ModernBERT embedding distance is below 0.05 but whose normalized resource demand distance exceeds 0.6, as well as semantically dissimilar queries with similar resource demand representations. These pairs explicitly encourage the encoder to distinguish semantic similarity from demand similarity and to preserve queries with similar resource requirements even when their surface semantics differ. The resulting hard pairs are incorporated into Stage 2 training to further shape the learned representation toward resource demand similarity rather than superficial semantic or length-based correlations.

The training process is shown in Algorithm 1, the encoder is trained progressively in two stages. We first fine-tune ModernBERT to learn the resource demand representation through demand regression, and then refine the representation by jointly optimizing demand regression and hard-pair contrastive learning.

B.2 Details of Routing Procedure and Model-Pair Configurations

B.2.1 Capability-Aware Cluster Filtering

We carefully prevent information leakage across all stages of training and evaluation. The more than 50K queries used to train the resource demand encoder are collected exclusively from WildChat and Brick Complexity Extractor, both of which are built from real-world user queries and responses. The 700 queries used in Stages 2–3 are constructed independently from Math, LeetCode, XpertBench, and MMLU, none of which overlaps with the datasets used for final evaluation. The evaluation set consists of LiveBench-Math (LBM (White et al., 2025)), MATH Arena (MAR (Balunovic et al., 2026)), OmniScience (OMN (Jackson et al., 2025)), MMLU-Pro (MMP (Wang et al., 2024)), GPQA-Diamond (GPD (Rein et al., 2023)), HLE (Phan et al., 2025), SciCode (SCI (Tian et al., 2024)), SWE-bench-Verified (SBV (Jimenez et al., 2024)), and IFBench (IFB (Pyatkin et al., 2026)). We additionally perform explicit duplicate checks when constructing the 700-query set to ensure that no evaluation question is reused in the routing data. To cover a broad range of routing scenarios, the 700 queries are deliberately sampled across domains and difficulty levels, ranging from relatively simple questions in Math and MMLU that smaller models can answer correctly to challenging problems from XpertBench on which even frontier models may fail. This design ensures that the routing queries provide diverse capability and resource demand requirements without exposing the router to any evaluation questions.

B.2.2 Cost-Aware Pairwise Ranking

Pairwise Cost Ranking MLP Training. We train a binary MLP classifier for each model pair listed in Table 10. The training data reuse the 700 queries from Stage 2, and for each query–model pair, we compute the realized cost as

$$\mathrm{Cost}(q,m) = P_m^{\mathrm{out}} T_{q,m}^{\mathrm{out}} + P_m^{\mathrm{reason}} T_{q,m}^{\mathrm{reason}},$$

Algorithm 1 Training the resource demand Encoder

Require: Demand dataset $\mathcal{D} = \{(q_i, \mathbf{d}_i)\}$, pair dataset $\mathcal{P} = \{(q_i, q_j, r_{ij})\}$, pretrained ModernBERT, number of demand dimensions $D = 24$
Ensure: Fine-tuned encoder f_θ and resource representation z_q
Initialize ModernBERT encoder f_θ from pretrained weights
Set maximum input length to 512 and use mean pooling
Attach a regression head with hidden dimension 256
Attach a projection head with hidden dimension 512 and output dimension 256
Normalize the projected representation before pairwise learning
Split $\mathcal{D}$ into training, validation, and test sets with $80\%/10\%/10\%$
Split $\mathcal{P}$ into training and validation sets with $80\%/20\%$
Compute the mean and standard deviation of each demand dimension using only the training split and apply z-score normalization
Stage 1: Demand Regression Warmup
Initialize AdamW with encoder learning rate 2×10^{-5} and head learning rate 1×10^{-4}
Apply layer-wise learning-rate decay with decay factor 0.90 to the encoder
for $epoch = 1$ to 7 **do**
  for each batch $\{(q_i, \mathbf{d}_i)\}$ **do**
    $z_i \leftarrow f_\theta(q_i)$
    $\hat{\mathbf{d}}_i \leftarrow g_{\text{reg}}(z_i)$
    $\mathcal{L}_{\text{reg}} \leftarrow \text{SmoothL1}(\hat{\mathbf{d}}_i, \mathbf{d}_i)$
    Update θ and regression-head parameters using $\mathcal{L}_{\text{reg}}$
  end for
  Evaluate on the validation set using MAE, RMSE, Spearman correlation, and Pearson correlation
  Save the checkpoint with the highest validation Spearman correlation
end for
Stage 2: Joint Demand and Pairwise Representation Learning
Initialize from the best Stage-1 checkpoint
Set regression weight $\lambda_r = 1.0$ and pairwise weight $\lambda_p = 0.3$
for $epoch = 1$ to 1 **do**
  for paired batches (q_i, q_j, r_{ij}) and demand batches **do**
    $z_i, z_j \leftarrow f_\theta(q_i), f_\theta(q_j)$
    $\mathcal{L}_{\text{reg}} \leftarrow \text{SmoothL1}(\hat{\mathbf{d}}, \mathbf{d})$
    $s_{ij} \leftarrow \cos(z_i, z_j)$
    $d_{ij} \leftarrow 1 - s_{ij}$
    if $r_{ij} = \text{close}$ **then**
      $\mathcal{L}_{\text{pair}} \leftarrow d_{ij}$
    else
      $\mathcal{L}_{\text{pair}} \leftarrow \max(0, 0.2 - d_{ij})$
    end if
    $\mathcal{L} \leftarrow \lambda_r \mathcal{L}_{\text{reg}} + \lambda_p \mathcal{L}_{\text{pair}}$
    Update θ and trainable head parameters using $\mathcal{L}$
  end for
  Evaluate demand prediction and pairwise representation quality
  Save the checkpoint with the highest validation Spearman correlation
end for
return Final resource demand encoder f_θ
=0

where P_m^{out} and P_m^{reason} denote the output-token and reasoning-token prices of model m, respectively. For each query, the model with the higher realized cost is assigned label 1, while the cheaper model is assigned label 0, forming a binary pairwise cost-ranking task. We randomly split the resulting data into training and test sets with an 8:2 ratio. Each classifier takes the resource demand vector produced by the resource demand encoder as input and predicts the relative cost of the two models.

Table 10: Model pairs used for pairwise cost ranking. Models are paired only when their official output prices differ by no more than $2\times$.

Model Pair	Price_1	Price_2	Ratio
Claude Fable 5 – GPT-6 Astra	\$50	\$50	1.00
Claude Opus 4.7 – Claude Opus 4.8	\$25	\$25	1.00
Claude Opus 4.7 – GPT-5.5	\$25	\$30	1.20
Claude Opus 4.8 – GPT-5.5	\$25	\$30	1.20
Claude Opus 4.7 – GPT-5.6-Sol	\$25	\$30	1.20
Claude Opus 4.8 – GPT-5.6-Sol	\$25	\$30	1.20
Claude Opus 5 – GPT-5.6-Sol	\$25	\$30	1.20
Gemini 3.1 Pro – GPT-5.6-Terra	\$12	\$12	1.00
Claude Sonnet 5 – Gemini 3.1 Pro	\$10	\$12	1.20
Claude Sonnet 5 – GPT-5.6-Terra	\$10	\$12	1.20
Gemini 3.1 Pro – Kimi K3	\$12	\$15	1.25
GPT-5.6-Terra – Kimi K3	\$12	\$15	1.25
Gemini 3.7 Flash – Gemini 3.8 Flash	\$3.75	\$3.75	1.00
Grok-4.5 – Qwen3.8 Max	\$6	\$6	1.00
Grok-4.5 – GLM-5.3	\$6	\$4.40	1.36
GLM-5.3 – Claude Haiku 4.5	\$4.40	\$5	1.14
Gemini 3.7 Flash – GLM-5.3	\$3.75	\$4.40	1.17
Gemini 3.8 Flash – GLM-5.3	\$3.75	\$4.40	1.17
GPT-5.6-Luna – DeepSeek V4 Flash	\$1.20	\$1.32	1.10
DeepSeek V4 Pro – GPT-5.6-Luna	\$0.87	\$1.20	1.38
Qwen3.8 Flash – GLM-5.3-Flash	\$0.47	\$0.50	1.06

We primarily construct pairwise predictors for models whose official prices differ by no more than $2\times$, as listed in Table 10. For models with an official price ratio between $2\times$ and $5\times$, although the higher-priced model is expected to incur a higher realized cost in most cases, query-dependent generation behavior can occasionally reverse this relationship. In particular, a model may generate substantially longer responses or consume a much larger reasoning budget, resulting in a realized cost that exceeds that of a nominally more expensive model. We therefore examine the 700-query execution data for such model pairs and additionally train a pairwise predictor whenever the cheaper model is observed to have a higher realized cost than the more expensive model on at least one query. For model pairs with an official price ratio exceeding $5\times$, we observe no cost-order reversals in the 700-query execution data, and do not construct pairwise predictors for such pairs. This selective construction allows the cost-ranking stage to capture empirically observed cost reversals without introducing unnecessary pairwise predictors for model pairs whose cost ordering remains effectively fixed.

To evaluate the effectiveness of the learned resource demand representation for cost ranking, we further compare it with the original ModernBERT representation under the same pairwise classification setting. Across multiple model pairs, the original ModernBERT achieves test accuracies ranging from 80.3% to 86.7%, whereas the resource demand representation achieves 93.5%-97.9%, demonstrating a substantial improvement in query-conditioned cost discrimination.

Inference-time Cost Ranking. Given a query q and the set of candidate models identified by Stage 2, Stage 3 performs fine-grained cost ranking among all capability-compatible models. The candidate set may contain models from multiple capability clusters and does not have a fixed size. We first perform iterative price-based pruning to remove clearly inefficient candidates. Specifically, if two remaining models have an official output-price ratio greater than $5\times$, we remove the more expensive model. This procedure is repeated until no pair of remaining models has an official price ratio exceeding $5\times$. Based on our empirical observation over the 700-query execution set, model pairs with a price ratio above $5\times$ do not exhibit cost-order reversals, making this pruning a safe simplification for the subsequent fine-grained ranking stage.

After price pruning, we construct the set of model pairs for which a trained pairwise predictor is available. Pairs with an official price ratio no greater than $2\times$ are directly covered by the pairwise predictors. For pairs with a price ratio between $2\times$ and $5\times$, a predictor is constructed only when

the 700-query training data contain at least one observed cost-order reversal, where the nominally cheaper model incurs a higher realized cost than the nominally more expensive model. If no such predictor exists for a remaining model pair, the pair is not evaluated at inference time. Therefore, the final pair set contains only comparisons for which query-conditioned cost variation has been empirically observed to be relevant.

The resource demand encoder is applied to the input query once to obtain its resource demand representation $\mathbf{z}_q$. The same representation is then fed into all available pairwise MLP classifiers associated with the remaining model pairs. For a model pair (m_i, m_j), the corresponding classifier outputs the probability that m_i has a higher realized cost than m_j:

$$p_{ij} = P\left(C(q, m_i) > C(q, m_j) \mid \mathbf{z}_q\right).$$

The complementary probability is

$$p_{ji} = 1 - p_{ij}.$$

We use these probabilities to construct a query-specific weighted directed cost-preference graph

$$\mathcal{G}_q = (\mathcal{V}_q, \mathcal{E}_q),$$

where each node represents a remaining candidate model. For each evaluated pair (m_i, m_j), the predicted cost relationship determines the direction of the corresponding edge, while the prediction probability provides its weight.

The resulting graph directly represents the query-conditioned relative cost structure among the candidate models. Because pairwise predictions are learned independently, their preferences may not always form a globally transitive ordering. For example, the classifiers may predict m_i to be more expensive than m_j, m_j to be more expensive than m_k, while simultaneously predicting m_k to be more expensive than m_i. Rather than imposing an artificial transitive ordering, we retain these pairwise predictions in the weighted directed graph and use their prediction probabilities to resolve such local inconsistencies. The final selected model is determined from the resulting graph by identifying the model with the strongest overall cost preference, i.e., the model that is consistently favored as the lower-cost option across the evaluated pairwise comparisons. This design allows Stage 3 to combine multiple query-conditioned pairwise judgments while preserving the confidence information provided by individual classifiers.

Table 11: Candidate resource demand dimensions collected from prior studies and LLM-assisted query analysis.

Candidate Dimensions			Candidate Dimensions		
Feature	**Source**	**Brief Description**	**Feature**	**Source**	**Brief Description**
Math Reasoning	Literature	Mathematical reasoning requirement	**Logic Reasoning**	LLM	Logical reasoning requirement
Coding Ability	LLM	Programming and coding requirement	**Domain Knowledge**	Literature	Domain knowledge requirement
Writing Ability	LLM	Writing and organization requirement	**Translation Ability**	LLM	Translation requirement
Planning Ability	LLM	Planning capability requirement	**Creative Thinking**	LLM	Creative thinking requirement
Task Type	Literature	Task category, such as classification, generation, QA, or coding	Reasoning Pattern	Literature	Reasoning pattern, such as multi-step or deductive reasoning
Complexity	Literature	Overall query complexity	Domain	Literature	Query domain or subject area
Cognitive Level	LLM	Cognitive depth required by the task	Cognitive Skill	LLM	Cognitive skill required by the query
Bottleneck Resource	Literature	Resource type most likely to limit completion	**Reasoning Depth**	Literature	Depth of reasoning required
Reasoning Steps	LLM	Number of reasoning steps required	**Verification Difficulty**	LLM	Difficulty of verifying intermediate results
Dependency Length	LLM	Length of dependencies across reasoning steps	**Search Space**	Literature	Complexity of the solution space
Proxy Solver	Literature	Signal from intermediate proxy-solver results	Iterative Reasoning	Literature	Need for iterative reasoning or refinement
Token Difficulty	Literature	Difficulty of individual generation steps	**Instruction Complexity**	LLM	Complexity of query instructions
Constraint Count	LLM	Number of explicit constraints	**Constraint Dependency**	Literature	Degree of coupling among constraints
Context Dependency	Literature	Dependence on contextual information	**Ambiguity**	LLM	Degree of ambiguity in the query
Syntactic Complexity	Literature	Syntactic complexity of the prompt	Semantic Complexity	Literature	Linguistic and semantic complexity
Keyword Signal	Literature	Informative keyword characteristics	Factual Signal	LLM	Factuality-related characteristics
Multi-Condition Complexity	Literature	Complexity from multiple conditions	Difficulty Density	Literature	Density of difficult requirements
Output Length	Literature	Expected response length	**Explanation Density**	LLM	Expected density of explanation
Formula Density	Literature	Expected density of formulas	**Code Density**	Literature	Expected density of code
Output Structure	LLM	Complexity of the required output structure	Length Prediction	Literature	Expected response length
Overall Resource	LLM	Overall resource requirement	Difficulty-Aware Budget	Literature	Compute budget based on query difficulty
Energy Requirement	Literature	Energy required for query execution	Semantic Embedding	LLM	Semantic information from query embeddings
Domain Embedding	Literature	Domain information from embeddings	IRT Difficulty	Literature	Latent difficulty from item-response behavior
IRT Features	Literature	Features for estimating latent difficulty	Option Plausibility	Literature	Plausibility of candidate answer choices
Query Ambiguity	Literature	Ambiguity in query wording or intent	Complexity Type	Literature	Query complexity type
Decomposition	Literature	Need to decompose the query	Disambiguation	Literature	Need to resolve ambiguity
Expansion	Literature	Need to expand the query	Abstraction	Literature	Need to abstract the query
Query Type	Literature	Type of task or query	Latent Task Type	Literature	Fine-grained latent task category
Difficulty-Aware Strategy	LLM	Strategy requirements induced by difficulty	Adaptive Perception	Literature	Required perception granularity
Test-Time Scaling	LLM	Compute scaling required by complexity	Problem Statement	LLM	Characteristics of the task description
Repository Context	Literature	Complexity of repository context	Solution Patch	Literature	Complexity of the required solution
Test Cases	Literature	Verification requirements from test cases	Programming Constructs	Literature	Complexity of programming constructs
Task Topic	Literature	Topic characteristics of programming tasks	Structural Decision Difficulty	Literature	Difficulty at key generation points
Task-Corpus Interaction	Literature	Interaction between task and corpus requirements			

C RESOURCE DEMAND ANNOTATION PROMPT

Prompt we use to generate resource demand labels

```
# Role

You are a professional Large Language Model Resource Analyzer.

Your task is not to answer the user's question, but to analyze:

> What capability resources, reasoning resources, and generation
    resources would a sufficiently capable and knowledgeable large
    language model need to invest to complete the task with high
    quality?

Always keep in mind:

You are analyzing the resource requirements of the task itself,
rather than whether the prompt is complex, long, or difficult to
    understand.

For example:

"Prove Fermat's Last Theorem"

Although the prompt contains only a few words, it requires extremely
     high mathematical capability, a very long reasoning chain, and
    extensive verification. Therefore, its resource requirement is
    extremely high.

In contrast,

"Output Hello World"

is also a one-sentence prompt, but it requires almost no reasoning
    resources.

Therefore:

Always analyze the task from the perspective of the resources
    required to complete it.

------------------------------------------------------------

# Workflow

For each query, strictly follow the following two stages.

==========================
Stage 1: Task Analysis
==========================

Do not assign scores immediately.

First, conduct the following analysis internally (do not output the
    analysis process).

Think through the following questions in order:

1 What does the user actually want to accomplish?

2 What capabilities are required to complete the task?
```

```
3 Does the task require substantial domain knowledge?

4 Does the task require complex reasoning?

5 Does the task require multi-step planning?

6 Does the task require continuous verification of intermediate
    results?

7 Are there many constraints that must be satisfied?

8 Does the task involve multiple subtasks?

9 Does the task require generating a long response?

10 Which resources are most likely to become bottlenecks?

After completing the above analysis, proceed to Stage 2.

------------------------------------------------------------

============================
Stage 2: Resource Scoring
============================

Based on the analysis in Stage 1, assign scores to all dimensions
    below.

All scores must use:

0 ~ 10

where:

0 = Almost no requirement

10 = Extremely high requirement

Make full use of the entire scoring range.

Do not concentrate all scores around 57 simply because of
    uncertainty.

For genuinely simple questions, confidently use:

0
1
2

For genuinely difficult questions, confidently use:

9
10

Different dimensions can be scored independently.

For example:

Mathematical requirement: 10

Coding requirement: 0
```

```
Writing requirement: 2

This is completely reasonable.

------------------------------------------------------------

# Scoring Guidelines

Very Low Resource Requirement (02)

Examples:

- Output Hello World
- Translate one sentence
- What is the weather today?
- Introduce Beijing
- Explain an ordinary concept

Characteristics:

Almost no complex reasoning;
No planning required;
Short output;
Limited capability requirements.

------------------------------------------------------------

Moderate Resource Requirement (35)

Examples:

- Write an email
- Summarize an article
- Write an ordinary Python function
- Explain the principles of Transformers
- Plan a one-week trip

Characteristics:

Requires some reasoning;
May require language organization;
Involves a small number of constraints;
Moderate output length.

------------------------------------------------------------

High Resource Requirement (68)

Examples:

- Design a database
- Implement a complex algorithm
- Develop a complete software design
- Develop a multi-module program
- Provide a relatively complex mathematical proof
- Design a long-form technical solution

Characteristics:

Requires sustained reasoning;
Involves multiple subtasks;
Requires planning;
Requires verification of intermediate steps;
```

```
Long output.

------------------------------------------------------------

Extremely High Resource Requirement (910)

Examples:

- Prove Fermat's Last Theorem
- Prove the Riemann Hypothesis
- Design a complete operating system
- Implement a large-scale compiler from scratch
- Provide a complete proof of a complex mathematical theory
- Design an extremely complex research methodology

Characteristics:

Extremely long reasoning chains;
Extensive verification;
Highly abstract;
Extremely high capability requirements;
Extremely high resource consumption.

------------------------------------------------------------

# Output Dimensions

Output the following Resource Profile.

==========================
I. Capability Requirement
==========================

math_reasoning
Mathematical reasoning requirement

logical_reasoning
Logical reasoning requirement

coding
Programming and coding requirement

knowledge
Domain knowledge requirement

writing
Writing and organization requirement

translation
Translation requirement

planning
Planning capability requirement

creative_thinking
Creative thinking requirement

------------------------------------------------------------

==========================
II. Reasoning Requirement
==========================
```

```
reasoning_depth
Reasoning depth

reasoning_steps
Reasoning chain length

verification_difficulty
Intermediate result verification requirement

dependency_length
Degree of dependency between steps

search_space
Complexity of the solution space

------------------------------------------------------------

==========================
III. Prompt Complexity
==========================

instruction_complexity
Instruction complexity

constraint_count
Number of constraints

constraint_dependency
Degree of coupling among constraints

context_dependency
Context dependency

ambiguity
Task ambiguity

------------------------------------------------------------

==========================
IV. Output Requirement
==========================

expected_output_length
Expected output length

expected_explanation_density
Explanation density

expected_formula_density
Formula density

expected_code_density
Code density

expected_structure_complexity
Output structure complexity

------------------------------------------------------------

==========================
V. Overall Assessment
==========================
```

```
overall_resource
Overall resource requirement

confidence
Confidence in the current scores

------------------------------------------------------------

# Output Requirements

Do not output any explanation.

Do not output any analysis process.

Do not output Markdown.

Do not output natural language.

Output JSON only.

Use the following format as a reference:


    {

        "resource_profile":{

            "math_reasoning":0,
            "logical_reasoning":0,
            "coding":0,
            "knowledge":0,
            "writing":0,
            "translation":0,
            "planning":0,
            "creative_thinking":0,

            "reasoning_depth":0,
            "reasoning_steps":0,
            "verification_difficulty":0,
            "dependency_length":0,
            "search_space":0,

            "instruction_complexity":0,
            "constraint_count":0,
            "constraint_dependency":0,
            "context_dependency":0,
            "ambiguity":0,

            "expected_output_length":0,
            "expected_explanation_density":0,
            "expected_formula_density":0,
            "expected_code_density":0,
            "expected_structure_complexity":0,

            "overall_resource":0,
            "confidence":0
        }
    }

I will provide you with several input questions and the response
    outputs of several LLMs below.
```