LONGCONTEXTROUTER: CONTEXT-AWARE MODEL ROUTING FOR LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Choosing a model for an LLM agent requires a delicate balance between task success and execution cost. However, standard routing methods are often myopic: they rely predominantly on the initial request, missing critical requirements buried in long-horizon task documents, prior messages, and execution records. We introduce LongContextRouter, a context-aware task-level router designed to capture this extended context through overlapping-window encoding and causal Transformer aggregation. This hierarchical representation interacts with learned candidate-model embeddings to predict task success, trained via pointwise and multi-positive listwise losses. A decoupled cost head allows model selection to dynamically reflect different cost preferences without requiring the success predictor to be retrained. Evaluated across four benchmarks and five candidate models—and leveraging historical execution text during training on SWE-bench Pro—LongContextRouter achieves 76.44% sample-average task success. This outperforms the strongest fixed-model baseline by 4.19 percentage points while simultaneously reducing recorded model-execution costs by 32.0%.

1 INTRODUCTION

Large language models enable agents to use tools and complete tasks through multiple steps (Yao et al., 2023). The model used by an agent affects both whether it can complete a task successfully and how much the execution costs. However, no single model is necessarily the best choice for every task: models differ in both their capabilities and their costs (Chen et al., 2024a). For some tasks, a less expensive model may already be sufficient, so using a more expensive one adds cost without improving the outcome. For other tasks, choosing a model solely because it is cheap may reduce the chance of success. Price alone is also an incomplete guide, since different models can have different strengths across tasks (Chen et al., 2024a). Model routing addresses this problem by selecting a model for each task rather than using the same model for all tasks, based on the information available when the choice is made (Ong et al., 2025). The goal is to choose a model that can complete the task successfully without spending more than necessary.

Existing routers learn selection policies from preference data or represent the compatibility between queries and candidate models. RouteLLM learns from preferences (Ong et al., 2025), while RouterDC, EmbedLLM, and GraphRouter use learned representations to capture differences in model suitability (Chen et al., 2024b; Zhuang et al., 2025; Feng et al., 2025). Many such methods center model selection on the current query (Ong et al., 2025; Chen et al., 2024b). In agent settings, however, relevant evidence may also appear in prior messages, tool observations, or records of earlier attempts (Yao et al., 2023; Zhang et al., 2026), while the query itself may contain lengthy task materials (Hu et al., 2026). MTRouter uses interaction histories for turn-level selection (Zhang et al., 2026), and Router-R1 coordinates model calls and incorporates returned responses into its decisions (Zhang et al., 2025). These developments motivate a representation question for context-aware routing: how should a router integrate information distributed across long task inputs and available context to assess candidate suitability? Long-document studies show that limiting input coverage can discard useful information and motivate hierarchical encoding (Dai et al., 2022; Pappagari et al., 2019). For routing, this means processing the full supplied input and learning how its chunks relate to one another; simply averaging their representations does not tell the router how those chunks are ordered.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

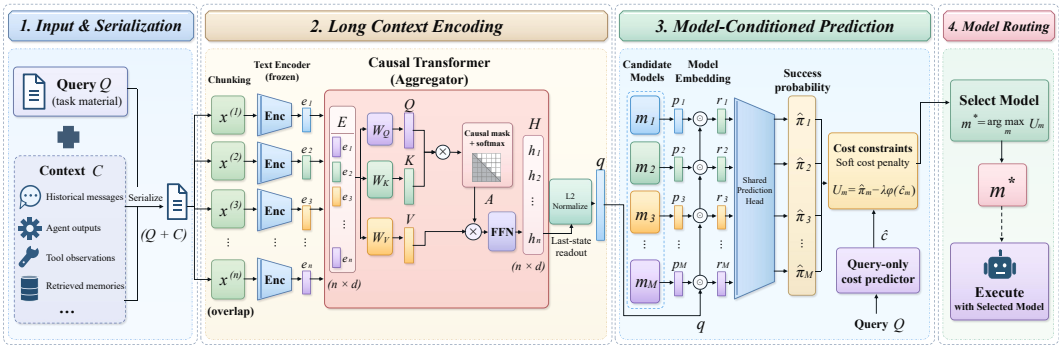

Figure 1: Overview of LongContextRouter. (1) The query $\boldsymbol{Q}$ and available context $\boldsymbol{C}$ are serialized in native agent format. On SWE-bench Pro, historical step text is used only during training; testing sets $\boldsymbol{C} = \varnothing$. (2) A shared frozen encoder maps overlapping chunks to embeddings. A position-aware causal Transformer aggregates them, and the last valid state is normalized to obtain $\boldsymbol{q}$. The attention query, key, and value matrices are distinct from the task query $\boldsymbol{Q}$; the diagram abstracts away positional encoding, residual connections, and intermediate normalization. (3) Each learned model embedding $\boldsymbol{p}_m$ interacts element-wise with $\boldsymbol{q}$, and a shared head estimates success probability $\hat{\pi}_m$. A separately fitted cost predictor uses frozen query-only router features. (4) The router selects the model with maximum cost-adjusted utility and uses it for the task. The diagram's soft penalty uses the benchmark-specific weight and scaling defined in Section 4.3, not a hard budget constraint. Pointwise and listwise training objectives are omitted from the inference flow.

In this work, we introduce **LongContextRouter**, a context-aware model routing framework. The router preserves the native format of task inputs and available context, encodes overlapping text windows with a shared frozen encoder, and combines their representations using a position-aware causal Transformer. Following hierarchical approaches to long-document encoding (Yang et al., 2016; Pappagari et al., 2019), this design handles inputs beyond a single encoder window without global attention over all input tokens. The combined input representation then interacts with a learned embedding for each candidate model to predict its probability of completing the task successfully. Pointwise and multi-positive listwise supervision train the router to select among candidates that can solve the task. A separate cost predictor allows the trade-off between task success and execution cost to be adjusted without retraining the success predictor.

We evaluate LongContextRouter on four benchmarks with five candidate models and 382 held-out tasks. It achieves 76.44% sample-average task success, exceeding the strongest fixed-model baseline by 4.19 percentage points while reducing recorded model-execution cost by 32.0% (Table 1). Controlled ablations examine historical training context, Transformer aggregation, listwise supervision, and cost awareness (Table 3). With Transformer aggregation and the cost-aware configuration fixed, adding historical training text improves sample-average success by 4.71 percentage points, although execution cost also increases. Replacing Transformer aggregation with mean pooling while retaining historical training context likewise lowers success at a lower cost. These results distinguish the gains over fixed-model deployment from the trade-offs among router variants. Reported costs cover selected-model execution and exclude router encoding and computation. Our contributions are:

- **Context-aware task-level routing.** We study model selection from native task inputs and available context, including the use of historical execution information during training.
- **Hierarchical input representation.** We use overlapping-window encoding and causal Transformer aggregation to represent inputs beyond one encoder window.
- **Model-conditioned routing.** We learn input–model compatibility with pointwise and multi-positive listwise supervision, and separate success prediction from cost preference.

2 Related Work

LLM routing. LLM routers select models according to their capabilities and costs. FrugalGPT estimates response quality within cost-aware cascades, whereas RouteLLM learns selection policies

from preference data (Chen et al., 2024a; Ong et al., 2025). Other methods learn representations to match queries with suitable models: RouterDC uses dual contrastive learning, EmbedLLM learns model embeddings, and GraphRouter represents task–query-model relations with a heterogeneous graph (Chen et al., 2024b; Zhuang et al., 2025; Feng et al., 2025). Our approach matches inputs with candidate models, but focuses on encoding long inputs and incorporating historical execution text.

History-conditioned routing. Router-R1 uses reinforcement learning to coordinate model calls and aggregate their responses (Zhang et al., 2025). MTRouter selects a model at each turn from the interaction history (Zhang et al., 2026). Our evaluation instead makes one choice per task; historical execution text on SWE-bench Pro augments training inputs rather than test-time histories.

Long-input representation. Hierarchical encoders combine local representations into a document-level representation. Hierarchical Attention Networks operate over words and sentences, while hierarchical Transformers encode segments before combining them with recurrent or Transformer layers (Yang et al., 2016; Pappagari et al., 2019). Sparse-attention methods such as BigBird reduce token-level attention complexity (Zaheer et al., 2020); Dai et al. (2022) compare sparse and hierarchical approaches and their document-splitting choices. LongContextRouter follows the hierarchical approach, using overlapping windows and position-aware aggregation to produce a representation for candidate-specific success prediction.

Additional training information. Learning using privileged information studies prediction with information available only during training (Vapnik & Izmailov, 2015). Distilling Step-by-Step uses generated rationales as supervision in multi-task training (Hsieh et al., 2023). Our SWE-bench Pro protocol shares this training–inference asymmetry, but adds execution text to router inputs and supervises candidate success rather than rationale generation.

3 Notation and Problem Formulation

Notation. Let $\mathcal{M} = \{m_1, \ldots, m_M\}$ be the candidate model pool. $\boldsymbol{Q}$ denotes the current query and attached task material, and $\boldsymbol{C}$ the available context, including prior messages, agent outputs, tool observations, and retrieved memories. Their native serialization is $\boldsymbol{x}$, with encoded representation $\boldsymbol{q} \in \mathbb{R}^d$. Candidate m has a learned embedding $\boldsymbol{p}_m \in \mathbb{R}^d$ and predicted success probability $\hat{\pi}_m$; $\boldsymbol{\theta}$ denotes router parameters. We use i for tasks and v for training views. Other symbols and dimensions are defined alongside the corresponding computations.

Problem Formulation. We select one model from $\mathcal{M}$ using information available before task execution. Native serialization and candidate success prediction are

$$\boldsymbol{x} = \mathrm{Native}(\boldsymbol{Q}, \boldsymbol{C}), \qquad f_{\boldsymbol{\theta}}(\boldsymbol{x}, m) \to \hat{\pi}_m. \tag{1}$$

On SWE-bench Pro, $\boldsymbol{Q}_i$ is the problem statement and $\boldsymbol{C}_v$ a pre-generated historical step output used only in training view v:

$$\boldsymbol{x}_v^{\mathrm{train}} = \mathrm{Native}(\boldsymbol{Q}_i, \boldsymbol{C}_v), \qquad \boldsymbol{x}_i^{\mathrm{test}} = \mathrm{Native}(\boldsymbol{Q}_i, \varnothing). \tag{2}$$

Context $\boldsymbol{C}_v$ need not contain a complete history. Label $y_{i,m}$ measures candidate m's full-task success from the beginning, not continuation from $\boldsymbol{C}_v$.

4 Method

LongContextRouter selects one model for each task based on the available input, predicted task success, and execution cost (Figure 1). A shared frozen encoder processes overlapping windows of the native serialized input, and a causal Transformer combines their embeddings into a compact representation (Section 4.1). This representation is combined with each model's learned embedding to predict its probability of success. Pointwise and multi-positive listwise losses train these predictions (Section 4.2). Once the success predictor is trained, we freeze the router and train a separate cost head using query-only features. At inference, the router subtracts a weighted predicted-cost penalty from each success score and selects the model with the highest adjusted score for the entire task (Section 4.3). The cost preference can be adjusted without retraining the success predictor.

4.1 LONG CONTEXT ENCODING

To ensure that the router processes all supplied context without window-level truncation, we separate local text encoding from cross-window aggregation, allowing the number of windows to grow with the input while keeping the final routing representation fixed-dimensional.

Window encoding. We split the serialized input $\boldsymbol{x}$ into overlapping chunks rather than truncate it to the encoder's context window. With chunk length L, overlap O, and stride S, the chunks and their embeddings are

$$\begin{aligned} S &= L - O, \qquad \boldsymbol{X} = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}], \\ \boldsymbol{e}_i &= \mathrm{TexEnc}(\boldsymbol{x}^{(i)}) \in \mathbb{R}^d, \qquad \boldsymbol{E} = [\boldsymbol{e}_1^\top; \ldots; \boldsymbol{e}_n^\top] \in \mathbb{R}^{n \times d}. \end{aligned} \tag{3}$$

Here each chunk contains at most L tokens, and n depends on the input length. We use $d = 1024$ in the reported configuration and stack chunk embeddings as rows of $\boldsymbol{E}$ in their serialized order. All chunks share a frozen encoder. Overlap reduces the risk of splitting useful evidence at window boundaries, and the resulting chunk sequence covers the full supplied input. The window length limits each encoder call, not the total input length. This hierarchy moves cross-window computation to the shorter sequence of chunk embeddings rather than applying global attention to all tokens. Each chunk is compressed into a single vector before aggregation.

Aggregation and readout. Independent chunk encoding does not model cross-chunk dependencies, and mean pooling does not explicitly represent chunk order. We therefore pass the ordered embeddings to a position-aware causal Transformer and read out its last valid position:

$$\begin{aligned} \boldsymbol{H} &= \mathrm{Agg}_{\boldsymbol{\theta}}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n) = [\boldsymbol{h}_1^\top; \ldots; \boldsymbol{h}_n^\top] \in \mathbb{R}^{n \times d}, \\ \boldsymbol{q}_{\mathrm{raw}} &= \mathrm{Norm}(\boldsymbol{h}_n) \in \mathbb{R}^d, \qquad \boldsymbol{q} = \frac{\boldsymbol{q}_{\mathrm{raw}}}{\|\boldsymbol{q}_{\mathrm{raw}}\|_2} \in \mathbb{R}^d. \end{aligned} \tag{4}$$

Each valid chunk attends only to itself and earlier chunks, with positional information indicating their order. The last valid position can therefore attend to all chunks and provides a d-dimensional readout, regardless of the number of chunks. We then apply ℓ_2 normalization to obtain a unit-length vector $\boldsymbol{q}$ without changing the direction of $\boldsymbol{q}_{\mathrm{raw}}$. With a single aggregation layer, the last valid position can attend to all chunks under either a causal or a bidirectional mask. Our ablations compare Transformer aggregation with mean pooling; they do not isolate the effect of causal masking.

4.2 MODEL-CONDITIONED ROUTING AND SELECTION-ALIGNED LEARNING

To translate the aggregated context representation into candidate-specific routing scores, we condition success prediction on model identity and combine individual outcome supervision with a selection objective that allows multiple successful candidates.

Candidate success prediction. Models can succeed on different tasks even when those tasks have similar difficulty. We capture this variation with a learned embedding $\boldsymbol{p}_m \in \mathbb{R}^d$ for each candidate m. Element-wise interaction with $\boldsymbol{q}$ produces a model-specific representation, which a shared linear head maps to failure and success logits:

$$\begin{aligned} \boldsymbol{r}_m &= \boldsymbol{q} \odot \boldsymbol{p}_m \in \mathbb{R}^d, \qquad \boldsymbol{z}_m = \boldsymbol{W}\boldsymbol{r}_m + \boldsymbol{b} \in \mathbb{R}^2, \\ s_m &= z_{m,1} - z_{m,0} \in \mathbb{R}, \qquad \hat{\pi}_m = \sigma(s_m) \in (0, 1). \end{aligned} \tag{5}$$

Here $z_{m,0}$ and $z_{m,1}$ are the failure and success scores. The interaction preserves dimension, $\boldsymbol{r}_m \in \mathbb{R}^d$, while $\boldsymbol{W} \in \mathbb{R}^{2 \times d}$ and $\boldsymbol{b} \in \mathbb{R}^2$ produce $\boldsymbol{z}_m \in \mathbb{R}^2$. Thus $s_m \in \mathbb{R}$ is a scalar margin and $\hat{\pi}_m \in (0, 1)$ a scalar probability for each of the M candidates. Each $\hat{\pi}_m$ estimates success independently; these probabilities need not sum to one across candidates. The candidate embeddings belong to the router, not to the underlying language models. Sharing the prediction head lets the router learn a common scoring rule while retaining candidate-specific interactions.

Pointwise supervision. Let $\mathcal{S}$ contain the supervised view–model pairs, with binary outcome $y_{v,m}$ and weight $w_{v,m}$. We fit each candidate's prediction to its observed outcome with weighted binary

cross-entropy:

$$\mathcal{L}_{\mathrm{BCE}} = \frac{\sum_{(v,m)\in\mathcal{S}} w_{v,m}\,\mathrm{BCE}(\hat{\pi}_{v,m}, y_{v,m})}{\sum_{(v,m)\in\mathcal{S}} w_{v,m}}. \tag{6}$$

Selection-aligned supervision. Several candidates may solve the same task. Rather than assign a unique correct model, we encourage the router to place probability mass on the set of successful candidates. For training view v, define

$$\mathcal{M}_v^+ = \{m \mid (v,m) \in \mathcal{S},\ y_{v,m} = 1\}, \qquad P_v(m) = \frac{\exp(s_{v,m}/\tau)}{\sum_{j=1}^{M} \exp(s_{v,j}/\tau)}. \tag{7}$$

Unlike the independent success estimates, P_v is a distribution over the candidate pool. We maximize its mass on $\mathcal{M}_v^+$ using

$$\mathcal{L}_{\mathrm{list}} = -\frac{1}{|\mathcal{V}^+|} \sum_{v\in\mathcal{V}^+} \log\left[\sum_{m\in\mathcal{M}_v^+} P_v(m)\right], \tag{8}$$

where $\mathcal{V}^+$ contains views with at least one successful candidate. The combined objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{BCE}} + \lambda_{\mathrm{list}}\mathcal{L}_{\mathrm{list}}. \tag{9}$$

BCE learns individual success estimates; the listwise term trains the choice among candidates without imposing an ordering on successful models.

4.3 Cost-Aware Inference

To encourage lower execution costs after routing while accounting for task success, we penalize predicted candidate costs during model selection. Success prediction and cost estimation are trained separately so that the cost preference can change without retraining the success predictor.

Separate cost-head training. Let g index a canonical cost-training group with query-only text $\boldsymbol{x}_g^Q$. Its fixed router features are

$$\boldsymbol{q}_g^Q = \mathrm{stopgrad}\big(F_{\boldsymbol{\theta}}(\boldsymbol{x}_g^Q)\big) \in \mathbb{R}^{1024}. \tag{10}$$

Here $F_{\boldsymbol{\theta}}$ includes the encoder and trained aggregator; its features are precomputed and held fixed. An affine head with no hidden layer or output activation predicts the candidates' log-costs:

$$\boldsymbol{\ell}_g = \boldsymbol{W}_c\boldsymbol{q}_g^Q + \boldsymbol{b}_c \in \mathbb{R}^M, \qquad \boldsymbol{W}_c \in \mathbb{R}^{M\times 1024}, \quad \boldsymbol{b}_c \in \mathbb{R}^M. \tag{11}$$

Each entry of $\boldsymbol{\ell}_g$ predicts one candidate's log-cost. We fit only $\boldsymbol{W}_c$ and $\boldsymbol{b}_c$ by minimizing mean squared error against $\log(1+c)$-transformed full-task training costs. At inference, an inverse transform and nonnegative clipping yield the task-specific cost vector $\widehat{\boldsymbol{c}}_g \in \mathbb{R}^M$. Appendix D gives the regression objective and clipping rule.

Cost scaling and selection. Benchmark identity t configures a scaling function ϕ_t and cost preference λ_t; it is not a predicted task category. SWE-bench Pro uses logarithmic scaling relative to a training-derived cost scale, whereas BBH, MemoryAgentBench, and MMAU use min–max scaling across predicted candidate costs. Logarithmic scaling avoids using the most expensive current candidate to set the scale. The exact formulas are in Appendix D. Neither cost prediction nor scaling uses realized test expenditure.

For benchmark t_g, the final selection is

$$U_{g,m} = \widehat{\pi}_m(\boldsymbol{x}_g^{\mathrm{test}}) - \lambda_{t_g}\phi_{t_g}(\widehat{\boldsymbol{c}}_g)_m, \qquad m_g^\star = \arg\max_{m\in\mathcal{M}} U_{g,m}. \tag{12}$$

The nonnegative λ_t values are cost-preference hyperparameters, not learned by cost regression; $\lambda_t = 0$ selects by success score alone.

Cost preference can change without retraining the success predictor, but this design does not guarantee calibrated success scores or cost predictions. Reported costs cover selected-model execution only; the end-to-end effect of router encoding and computation overhead remains unevaluated.

Table 1: Main comparison on the same five-model pool and 382 held-out tasks. MAB denotes MemoryAgentBench, and MMAU (SC) denotes the Self-correction subset. Accuracy is reported as a percentage and cost in USD. Params. reports total / activated parameters (B: billion; T: trillion). Models with published parameter counts are ordered from largest to smallest; GLM's author-supplied total is listed separately, and — denotes an unspecified activated count. AVG is the fixed-model mean. Max Expert and Oracle use held-out outcomes and are non-deployable references. Bold aggregate accuracies identify the strongest deployable configuration.

Group	Model / Variant	Params. total / active	BBH		MAB		MMAU (SC)		SWE-bench Pro		Dataset Avg. Acc.	Sample Avg. Acc.	Total Cost
			Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost			
Model pool	DeepSeek-V4-Pro[a]	1.6T / 49B	97.00	0.05	62.00	0.25	60.98	0.06	67.00	16.77	71.74	72.25	17.13
	DeepSeek-V4-Pro[b]	1.6T / 49B	98.00	0.18	55.00	1.61	50.00	0.07	45.00	16.16	62.00	62.57	18.02
	DeepSeek-V4-Flash[c]	284B / 13B	94.00	0.02	62.00	0.33	63.41	0.02	66.00	2.99	71.35	71.73	3.35
	Qwen3.5[d]	122B / 10B	97.00	0.51	69.00	8.29	52.44	0.11	51.00	535.08	67.36	68.06	544.00
	GLM-4-Flash[e]	30B / —	55.00	0.00	24.00	0.00	3.66	0.00	1.00	0.00	20.91	21.73	0.00
Reference	AVG	—	88.20	0.15	54.40	2.10	46.10	0.05	46.00	114.20	58.67	59.27	116.50
	Random Router	—	89.00	0.14	50.00	1.97	54.88	0.06	37.00	76.57	57.72	57.85	78.74
	EmbedLLM	—	98.00	0.11	63.00	1.68	59.76	0.07	67.00	90.50	71.94	72.51	92.36
	Max Expert	—	98.00	0.18	69.00	8.29	63.41	0.02	67.00	16.77	74.35	74.87	25.26
	Oracle	—	100.00	0.01	78.00	0.65	74.39	0.01	86.00	23.33	84.60	85.08	24.00
Router	**LongContextRouter**	—	97.00	0.06	67.00	0.37	60.98	0.06	78.00	11.16	**75.74**	**76.44**	11.65

[a]DeepSeek-V4-Pro-0813 (2026-08-13). [b]DeepSeek-V4-Pro Preview (2026-04-24) (DeepSeek-AI, 2026). [c]DeepSeek-V4-Flash-0731 (2026-07-31); Its architecture and size are unchanged from the preview. [d]Qwen3.5-122B-A10B (Qwen Team, 2026). [e]GLM-4-Flash-250414. DeepSeek and Qwen parameter counts follow published specifications. GLM's 30B total is author-supplied, not established by the official endpoint documentation; its activated count is unspecified.

5 Experiments

We evaluate LongContextRouter on BBH, MMAU Self-correction, MemoryAgentBench, and SWE-bench Pro, comparing task success and execution cost against fixed models and routing baselines. Ablations examine historical training context, aggregation, training objectives, and cost awareness.

5.1 Experimental Setup

Benchmarks and protocol. We evaluate on BBH (Suzgun et al., 2023), MMAU Self-correction (Yin et al., 2025), MemoryAgentBench (MAB) (Hu et al., 2026), and SWE-bench Pro (Deng et al., 2026), with 867 training, 233 validation, and 382 test routing groups. Splits are task-disjoint, keeping each task's records and views together. On SWE-bench Pro, full-input training includes historical step text; query-only training and testing use only the problem statement. Labels measure full-task success from the beginning. All comparisons share test tasks, candidate outcomes, and cost accounting. Appendix E reports detailed splits and cost configurations.

Models and Metrics. Appendix A lists the five-model pool, versions, and reference prices. **Acc.** measures task success (%), defined as F1 > 0.5 on MAB and task resolution on SWE-bench Pro. **Dataset-Avg Acc.** and **Sample-Avg Acc.** average equally over benchmarks and test instances, respectively. **Cost** is recorded full-task candidate-model expenditure per benchmark in USD; **Total Cost** sums across benchmarks. Router encoding, computation, and hosting are excluded.

Implementation. We use a frozen Qwen3-Embedding-0.6B encoder with 32,768-token windows, a 512-token overlap, and no global truncation, followed by a one-layer causal Transformer. Model selection is based on the best checkpoint determined on the validation set. Additional details on the experimental protocol, training procedure, and deployment setup are provided in Appendix C.

5.2 Main Results

We compare LongContextRouter with five **fixed-model baselines**, each assigning all tasks to one candidate, a **Random Router**, and an adapted **EmbedLLM** baseline (Zhuang et al., 2025). We adapt EmbedLLM to the same five-model pool, task-start inputs, and recorded full-task success targets. **AVG** is the fixed-model mean, not a routing policy. Two non-deployable references use held-out

outcomes: **Max Expert** selects the best fixed candidate per benchmark, while **Oracle** selects a successful candidate whenever one exists. Table 1 reports these comparisons; Oracle bounds accuracy within the recorded pool, not minimum cost.

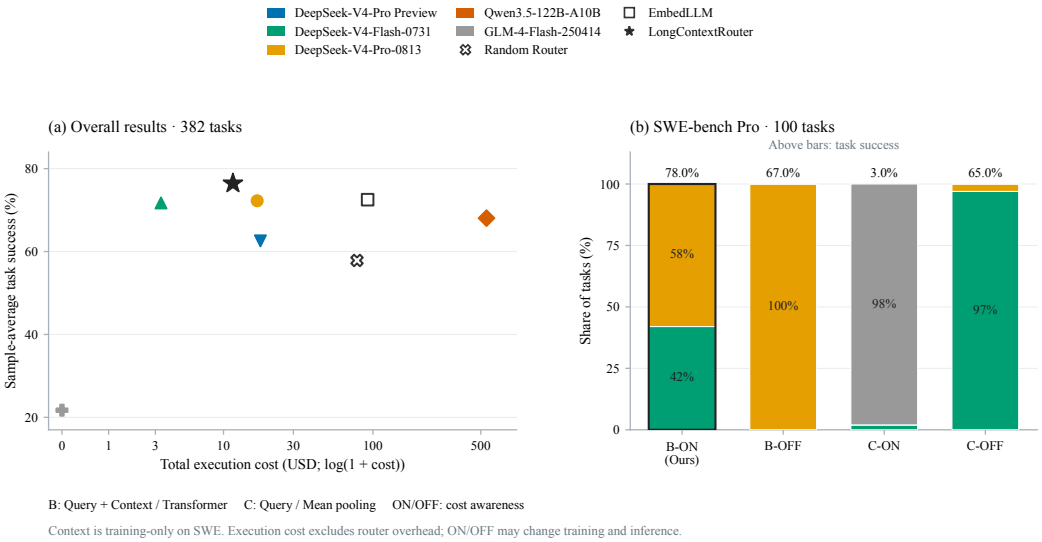

Figure 2: Accuracy, execution cost, and model selection. (a) Aggregate results on 382 held-out tasks across four benchmarks; each scatter point is a reported configuration, not a point on a fixed-checkpoint cost-preference sweep. The cost axis uses $\log(1+c)$ with tick labels in USD and includes only candidate-model execution, excluding router overhead. (b) Routing distributions for 100 SWE-bench Pro tasks. Each task contributes one model choice. Percentages inside bars denote selection shares; percentages above bars denote task success. B uses Query + Context with Transformer aggregation; C uses Query with mean pooling. Context is training-only on SWE-bench Pro. ON/OFF changes training and inference cost settings and does not isolate the inference penalty.

Overall performance. LongContextRouter improves both aggregate success and execution cost relative to the strongest fixed model, DeepSeek-V4-Pro. It reaches 75.74% dataset-average and 76.44% sample-average success at $11.65: gains of 4.00 and 4.19 percentage points with a 32.0% cost reduction. It also exceeds Random Router by 18.59 sample-average points at 85.2% lower cost, and Max Expert by 1.57 points at 53.9% lower cost. Compared with adapted EmbedLLM, LongContextRouter improves dataset-average and sample-average accuracy by 3.80 and 3.93 percentage points, respectively, while reducing execution cost from $92.36 to $11.65 (87.4%). Accuracy increases by 4.00 points on MAB, 1.22 on MMAU, and 11.00 on SWE-bench Pro, but decreases by 1.00 point on BBH. These results compare complete routing configurations at their reported operating points and do not isolate the contribution of context encoding. Figure 2(a) visualizes these aggregate operating points: LongContextRouter achieves the highest sample-average success among the compared deployable configurations, while DeepSeek-V4-Flash provides a lower-cost alternative.

Differences Across Benchmarks. Routing benefits differ across benchmarks. On SWE-bench Pro, success increases from the best fixed model's 67.00% to 78.00%, while cost decreases from $16.77 to $11.16. This benchmark accounts for 95.8% of the router's total cost and therefore largely determines its overall cost–performance profile. On MAB, the router achieves 67.00% success at $0.37, two points below Qwen's best fixed-model result but at 95.5% lower cost. On BBH and MMAU, it trails the best fixed model by 1.00 and 2.43 points, respectively. Flash offers a cheaper alternative at $3.35 and 71.73% sample-average success. Oracle reaches 85.08% sample-average success, exceeding LongContextRouter by 8.64 points and indicating remaining performance headroom within the current pool.

Table 2: Input, aggregation, and cost-awareness ablations. Accuracy is reported as a percentage and execution cost in USD. Bold marks the highest aggregate accuracies among the eight settings.

Training input / Aggregation	Cost setting	BBH		MAB		MMAU (SC)		SWE-bench Pro		Dataset-Avg Acc.	Sample-Avg Acc.	Total Cost
		Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost			
A: Query Transformer	ON	95.00	0.06	62.00	0.33	62.20	0.02	66.00	2.99	71.30	71.73	3.40
	OFF	98.00	0.11	63.00	1.68	59.76	0.07	67.00	90.50	71.94	72.51	92.36
B (ours): Query + Context Transformer	ON	97.00	0.06	67.00	0.37	60.98	0.06	78.00	11.16	**75.74**	**76.44**	11.65
	OFF	97.00	0.06	68.00	0.89	60.98	0.06	67.00	16.77	73.24	73.82	17.78
C: Query Mean pooling	ON	94.00	0.02	62.00	0.33	63.41	0.02	3.00	0.04	55.60	55.24	0.41
	OFF	93.00	0.03	62.00	0.33	63.41	0.02	65.00	3.42	70.85	71.20	3.79
D: Query + Context Mean pooling	ON	94.00	0.02	62.00	0.33	63.41	0.02	66.00	2.99	71.35	71.73	3.35
	OFF	94.00	0.02	62.00	0.33	63.41	0.02	65.00	3.53	71.10	71.47	3.90

5.3 Input, Aggregation, and Cost Awareness

To complement the main comparison, Table 2 crosses training input, aggregation, and cost awareness on the same model pool and test tasks. On SWE-bench Pro, Context denotes pre-generated historical step text used only during training; all settings use only the problem statement at test time. Cost OFF disables training and inference cost terms.

Training Input and Aggregation. With cost awareness enabled (ON), incorporating historical training context (B vs. A) and using Transformer aggregation (B vs. D) each improves sample-average accuracy by 4.71 percentage points. The context comparison holds aggregation fixed, while the aggregation comparison uses the same query-plus-context training inputs. However, B incurs higher execution costs than both A and D, so the gains reflect an accuracy–cost trade-off rather than an improvement at matched cost.

When cost awareness is disabled (OFF), historical training context and Transformer aggregation still improve sample-average accuracy by 1.31 and 2.35 percentage points, respectively. Their larger gains under cost ON suggest that the benefits of these components depend on the cost-aware routing configuration.

Interaction with Cost Awareness. Configuration B achieves the highest overall accuracy among all eight settings when cost awareness is enabled. Compared with its cost-unaware counterpart, its sample-average accuracy increases from 73.82% to 76.44%, while total cost decreases from $17.78 to $11.65, corresponding to a 34.5% reduction. Thus, B-ON improves both metrics relative to B-OFF at the reported operating points.

Figure 2(b) shows the corresponding model-selection distributions on SWE-bench Pro. B-ON assigns 42% of tasks to DeepSeek-V4-Flash-0731 and 58% to DeepSeek-V4-Pro-0813, whereas B-OFF selects Pro-0813 for all tasks. C-ON assigns 98% of tasks to GLM-4-Flash and achieves only 3% task success; C-OFF assigns 97% to Flash and reaches 65%. These distributions help explain the observed accuracy–cost trade-offs, but do not isolate the inference-time cost penalty because the configurations also differ in training settings. Full routing distributions appear in Appendix B. Section 5.5 further examines training objectives under a fixed inference policy.

5.4 Component Ablation

Table 3 summarizes the four component ablations. Input, aggregation, and cost interactions are analyzed in Section 5.3; we focus here on listwise supervision.

Listwise Supervision. Listwise supervision improves sample-average success from 72.25% to 76.44%, a gain of 4.19 percentage points. It improves BBH, MAB, and SWE-bench Pro by 1.00, 5.00, and 12.00 points, respectively, while reducing MMAU by 2.43 points. These aggregate gains come with higher execution cost ($11.65 versus $3.38). Section 5.5 further compares different objectives.

Table 3: Component ablation on the same test split. MAB denotes MemoryAgentBench. Historical-context removal and Transformer replacement correspond to A and D with cost ON in Table 2, respectively; each retains the other component. Cost OFF disables training and inference cost terms. Other rows retain the main inference policy.

Group	Model / Variant	BBH		MAB		MMAU (SC)		SWE-bench Pro		Dataset Avg. Acc.	Sample Avg. Acc.	Total Cost
		Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost			
Router	**LongContextRouter (Full)**	97.00	0.06	67.00	0.37	60.98	0.06	78.00	11.16	**75.74**	**76.44**	11.65
Ablation	w/o Historical training context	95.00	0.06	62.00	0.33	62.20	0.02	66.00	2.99	71.30	71.73	3.40
	w/o Transformer aggregation (mean pooling)	94.00	0.02	62.00	0.33	63.41	0.02	66.00	2.99	71.35	71.73	3.35
	w/o Listwise supervision	96.00	0.05	62.00	0.33	63.41	0.02	66.00	2.99	71.85	72.25	3.38
	w/o Cost awareness	97.00	0.06	68.00	0.89	60.98	0.06	67.00	16.77	73.24	73.82	17.78

Table 4: Training objective ablation under a fixed inference policy. Each component is evaluated relative to the baseline configuration. MAB denotes MemoryAgentBench.

Group	Model / Variant	BBH		MAB		MMAU (SC)		SWE-bench Pro		Dataset Avg. Acc.	Sample Avg. Acc.	Total Cost
		Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost			
Training loss	Base (BCE + cost ranking)	96.00	0.05	62.00	0.33	63.41	0.02	66.00	2.99	71.85	72.25	3.38
	+ Decision-aligned loss	95.00	0.08	64.00	0.66	58.54	0.04	54.00	106.23	67.88	68.32	107.01
	+ Task-aux classification loss	93.00	0.03	66.00	0.36	63.41	0.03	57.00	12.61	69.85	70.16	13.03
	+ Listwise + Decision-aligned	94.00	0.05	65.00	0.29	60.98	0.05	52.00	52.02	67.99	68.32	52.41
	+ Listwise + Task-aux	98.00	0.18	67.00	1.06	58.54	0.06	63.00	12.49	71.63	72.25	13.80
Router	**LongContextRouter**	97.00	0.06	67.00	0.37	60.98	0.06	**78.00**	11.16	**75.74**	**76.44**	11.65

5.5 Training Objective Ablation

Table 4 compares additional training objectives under a fixed inference policy.

Auxiliary objectives have uneven effects across benchmarks. Decision-aligned and task-auxiliary losses improve MAB from 62.00% to 64.00% and 66.00%, respectively, but reduce SWE-bench Pro from 66.00% to 54.00% and 57.00%. These improvements therefore do not translate into higher aggregate accuracy.

Combining listwise and task-auxiliary supervision remains competitive on BBH and MAB, but yields 63.00% success on SWE-bench Pro and 72.25% sample-average success, versus 78.00% and 76.44% for LongContextRouter. The results support listwise supervision without additional auxiliary losses in this comparison, but do not establish robustness across random seeds.

6 Conclusion

LongContextRouter selects models for agent tasks by combining hierarchical long-context encoding, candidate-specific success prediction, and cost-aware routing. Across four benchmarks, it achieves a sample-average task success rate of 76.44% while reducing the recorded execution cost relative to the strongest fixed-model baseline. Ablation studies further demonstrate that historical execution context, Transformer-based cross-window aggregation, listwise supervision, and cost awareness jointly contribute to effective model selection across heterogeneous agent tasks. In particular, LongContextRouter shows substantial gains on long-context and multi-step settings such as MemoryAgentBench and SWE-bench Pro, highlighting the value of historical execution text as additional training information. These results suggest that historical execution text can improve task-level routing when used during training, even when test-time routing uses only the problem statement, as in our SWE-bench Pro evaluation. Future work will extend the framework to richer decision-time histories and step-level routing while incorporating end-to-end routing overhead into the optimization objective.

AI USE STATEMENT

Generative AI tools were used to assist with literature discovery, language editing, and preparation of an early manuscript draft under author supervision. The authors are responsible for verifying all technical claims, citations, experimental results, and released artifacts.

REFERENCES

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

Shuhao Chen, Weisen Jiang, Baijiong Lin, James T. Kwok, and Yu Zhang. RouterDC: Query-based router by dual contrastive learning for assembling large language models. In *Advances in Neural Information Processing Systems*, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7a641b8ec86162fc875fb9f6456a542f-Abstract-Conference.html.

Xiang Dai, Ilias Chalkidis, Sune Darkner, and Desmond Elliott. Revisiting transformer-based models for long document classification. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 7212–7230. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-emnlp.534. URL https://aclanthology.org/2022.findings-emnlp.534/.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. Technical report, DeepSeek-AI, 2026. URL https://arxiv.org/abs/2606.19348.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean M. Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks? In *Forty-third International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/61047.

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A graph-based router for LLM selections. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/41b6674c28a9b93ec8d22a53ca25bc3b-Abstract-Conference.html.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *International Conference on Learning Representations*, volume 2026, pp. 156259–156291, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/fd1eff9dd295df50a41f2521942fa31d-Abstract-Conference.html.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

Raghavendra Pappagari, Piotr Zelasko, Jesus Villalba, Yishay Carmiel, and Najim Dehak. Hierarchical transformers for long document classification. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 838–844. IEEE, December 2019. doi: 10.1109/ASRU46091.2019.9003958. URL https://doi.org/10.1109/ASRU46091.2019.9003958.

Qwen Team. Qwen3.5-122B-A10B model card, February 2026. URL https://huggingface.co/Qwen/Qwen3.5-122B-A10B.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Vladimir Vapnik and Rauf Izmailov. Learning using privileged information: Similarity control and knowledge transfer. *Journal of Machine Learning Research*, 16(61):2023–2049, 2015. URL https://www.jmlr.org/papers/v16/vapnik15b.html.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489. Association for Computational Linguistics, 2016. doi: 10.18653/v1/N16-1174. URL https://aclanthology.org/N16-1174/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Guoli Yin, Haoping Bai, Shuang Ma, Feng Nan, Yanchao Sun, Zhaoyang Xu, Shen Ma, Jiarui Lu, Xiang Kong, Aonan Zhang, Dian Ang Yap, Yizhe Zhang, Karsten Ahnert, Vik Kamath, Mathias Berglund, Dominic Walsh, Tobias Gindele, Juergen Wiest, Zhengfeng Lai, Xiaoming Simon Wang, Jiulong Shan, Meng Cao, Ruoming Pang, and Zirui Wang. MMAU: A holistic benchmark of agent capabilities across diverse domains. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 4752–4780. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.267. URL https://aclanthology.org/2025.findings-naacl.267/.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

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*, 2025. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/ceaa137fce916aba5c65fceb1309088b-Abstract-Conference.html.

Yiqun Zhang, Hao Li, Zihan Wang, Shi Feng, Xiaocui Yang, Daling Wang, Bo Zhang, Lei Bai, and Shuyue Hu. MTRouter: Cost-aware multi-turn LLM routing with history–model joint embeddings. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44206–44226, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.2045. URL https://aclanthology.org/2026.acl-long.2045/.

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

A Model Sizes, Versions, and Reference Prices

Table 5 summarizes candidate model sizes and official token prices in USD. DeepSeek and Qwen parameter counts follow the published language-model specifications (DeepSeek-AI, 2026; Qwen Team, 2026). The Flash update retains the preview's architecture and size. For GLM-4-Flash, we report an author-supplied total of 30B parameters, with the activated parameter count unspecified. Exact versions and pricing conditions appear below the table.

Table 5: Candidate sizes and official USD prices per million tokens. Params. reports total / activated parameters (B: billion; T: trillion); — denotes an unspecified activated count. Rows are ordered by total size. Input prices are for uncached tokens. Flash-0731 and Pro-0813 use off-peak rates; Pro Preview uses its launch list price. Version and pricing conditions are specified below.

LLM	Params. total / active	Input ($/1M)	Output ($/1M)
GLM-4-Flash[e]	30B / —	0.00	0.00
Qwen3.5[a]	122B / 10B	0.115	0.917
DeepSeek-V4-Flash[b]	284B / 13B	0.22	0.66
DeepSeek-V4-Pro[c]	1.6T / 49B	1.74	3.48
DeepSeek-V4-Pro[d]	1.6T / 49B	0.66	1.98

[a]Qwen3.5-122B-A10B: Alibaba Cloud Model Studio, Beijing, input length $\leq$ 128K, using the USD prices in the official English documentation. [b]DeepSeek-V4-Flash-0731 (2026-07-31). [c]DeepSeek-V4-Pro Preview (2026-04-24): launch list prices. [d]DeepSeek-V4-Pro-0813 (2026-08-13). [e]GLM-4-Flash-250414: free input and output.

These are the providers' published USD rates. Experimental tables retain recorded full-task expenditure and are not recalculated from this table. Reconstructing recorded costs also requires each run's billing date, region, cache usage, discounts, and conversion convention.

B Routing Selection Distributions

Figure 3 reports the candidate-model selection shares for all eight configurations on each benchmark, using the held-out test splits summarized in Table 6.

C Additional Experimental Details

Benchmark composition and inputs. BBH evaluates logical reasoning, MMAU uses its Self-correction subset, MAB evaluates single-pass long-context processing, and SWE-bench Pro contains multi-step repository-level coding-agent tasks. Dataset splits are defined at the original routing-group level, as summarized in Table 6. BBH contains 392/108/100 training, validation, and test groups; MAB contains 155/45/100; MMAU contains 160/40/82; and SWE-bench Pro contains 160/40/100. This yields 867 training groups, 233 validation groups, and 382 test tasks in total. For context-based training on SWE-bench Pro, the training and validation routing groups are expanded into 640 and 160 model-specific embedding views, respectively, while test-time routing remains query-only. Inputs otherwise follow Section 3, with BBH, MMAU, and MAB retaining their benchmark-provided task material.

D Cost Prediction and Scaling Details

We expand the regression and scaling steps used in Section 4.3. For cost-training group g, the frozen query-only features and affine prediction head are defined in Equations 10 and 11.

Regression and prediction. The output $\boldsymbol{\ell}_g \in \mathbb{R}^M$ contains one log-cost prediction per candidate; $\boldsymbol{b}_c \in \mathbb{R}^M$ is the corresponding bias. For recorded full-task costs $c_{g,m}$ in USD, we optimize only $\boldsymbol{W}_c$ and $\boldsymbol{b}_c$ by minimizing mean squared error in $\log(1+c)$ space:

$$\mathcal{L}_{\text{cost}} = \frac{1}{|\mathcal{G}_{\text{train}}|M} \sum_{g \in \mathcal{G}_{\text{train}}} \sum_{m=1}^{M} \left[\ell_{g,m} - \log(1 + c_{g,m})\right]^2 . \tag{13}$$

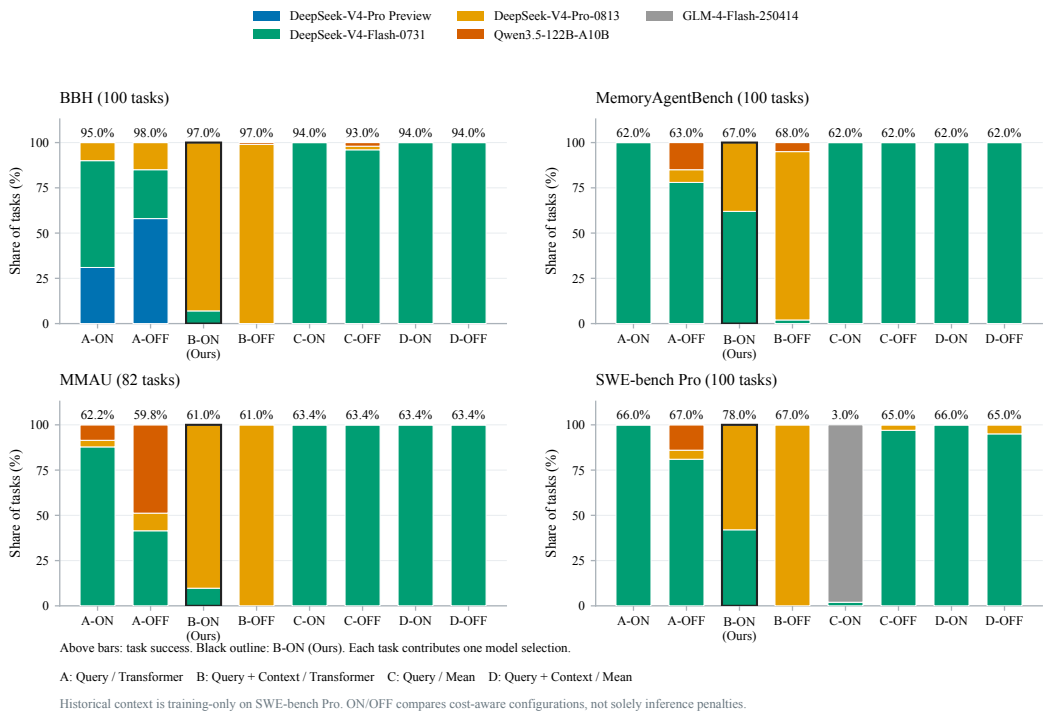

Figure 3: Complete model-selection distributions across benchmark test sets. Bar segments show the fraction of tasks assigned to each candidate model, and numbers above the bars indicate task success rates. The black outline identifies B-ON (LongContextRouter). A/B use Transformer aggregation, whereas C/D use mean pooling; A/C use Query and B/D use Query + Context during training. For SWE-bench Pro, context-based variants use execution context during training, while test-time routing uses query-only inputs. ON/OFF denotes whether cost awareness is enabled.

Only the cost-head parameters are updated in this regression stage. The task-auxiliary classification loss in our ablations is a separate objective.

At inference, query-only features produce predictions mapped back to the cost scale:

$$\hat{c}_{g,m} = \mathrm{clip}(\mathrm{expm1}(\ell_{g,m}), 0, \exp(20)) \,. \tag{14}$$

Here $\ell_{g,m}$ is the mth entry of $\boldsymbol{\ell}_g$, and $\widehat{\boldsymbol{c}}_g = (\hat{c}_{g,1}, \ldots, \hat{c}_{g,M})^\top$ collects the predictions. This instance-specific vector replaces task-average costs in routing. Realized test execution costs only measure expenditure; they are never cost-head or selection inputs.

Cost scaling. Benchmark identity t configures scaling and preference as a conditioning index, not a task-type prediction. Candidate mean full-task costs are computed per benchmark using training groups only:

$$\bar{c}_{t,m} = \frac{1}{|\mathcal{G}_{\mathrm{train},t}|} \sum_{g \in \mathcal{G}_{\mathrm{train},t}} c_{g,m}, \qquad a_t = \mathrm{median}\left\{\bar{c}_{t,m} : \bar{c}_{t,m} > 0\right\}. \tag{15}$$

The scale a_t is the median of positive candidate-wise training means. For the current query, the predicted-cost minimum and range are

$$\hat{c}_g^{\min} = \min_j \hat{c}_{g,j}, \qquad d_g = \max_j \hat{c}_{g,j} - \hat{c}_g^{\min}. \tag{16}$$

SWE-bench Pro uses the training-median logarithmic scale, whereas BBH, MemoryAgentBench, and MMAU use min–max scaling:

$$\phi_t(\widehat{\boldsymbol{c}}_g)_m = \begin{cases} \log(1 + \hat{c}_{g,m}/a_t) - \log\left(1 + \hat{c}_g^{\min}/a_t\right), & t = \text{SWE-bench Pro}, \\ \left(\hat{c}_{g,m} - \hat{c}_g^{\min}\right)/d_g, & t \neq \text{SWE-bench Pro}. \end{cases} \tag{17}$$

These expressions apply when $a_t > 0$ and $d_g > 0$, respectively. These scales require no realized test costs: a_t comes from training records, while the minimum and range come from predictions. Unlike range-based scaling, the logarithmic scale uses the training-derived a_t rather than the highest current predicted cost to set its scale. This reduces the compression of cheaper models' relative penalties when one candidate is much more expensive.

E Dataset Splits and Cost Configurations

Table 6 summarizes the group-level dataset splits, benchmark-specific cost-preference hyperparameters λ_t, and cost-scaling rules. All records and derived views of the same task remain in one partition. For SWE-bench Pro, historical context expands the 160 training and 40 validation groups into 640 and 160 model-specific views, respectively, without increasing the number of independent tasks. The 100 test inputs remain query-only. The reference median used for SWE-bench Pro cost scaling is computed from training data only.

Table 6: Dataset splits and benchmark-specific cost configurations.

Benchmark	**BBH**	**MAB**	**MMAU (SC)**	**SWE-bench Pro**
Train / Val / Test	392 / 108 / 100	155 / 45 / 100	160 / 40 / 82	160 / 40 / 100
Selected λ_t	0.0	2.0	0.1	1.0
Cost Scale	Min–Max	Min–Max	Min–Max	Train Log1p Median

Note: Split sizes are reported at the original routing-group level. For context-based training on SWE-bench Pro, each routing group is expanded into model-specific views, resulting in 640 training and 160 validation embedding views. The test set remains unchanged with 100 query-only inputs.