BEYOND FUNCTIONAL RELEVANCE: BENCHMARKING COST-AWARE SKILL RETRIEVAL FOR EFFICIENT AGENT EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

As the scale of reusable Skills continues to grow, Skill Retrieval has become an important mechanism for LLM Agents to access task-relevant knowledge and operational guidance. Existing retrieval methods primarily rely on functional relevance, focusing on identifying task-supporting Skills. However, functional relevance alone may be insufficient for selecting Skills that support efficient downstream execution: viable Skills for the same task may guide Agents toward different technical approaches, resulting in substantially different downstream execution costs. To investigate this issue, we introduce **Cost-Aware Skill Retrieval** and construct the **Cost-Aware Skill Retrieval Benchmark**. Experiments on this benchmark support this concern, showing that strong recall of task-supporting Skills does not necessarily translate into a consistent preference for more execution-efficient Skills. Motivated by this finding, we develop a lightweight **Cost-Aware Reference Method** that uses a two-stage process to improve cost-aware Skill selection. Experimental results indicate that the method can favor lower-cost Skills and translate this preference into actual downstream savings, reducing Agent execution time by up to 51.55% and context-adjusted token consumption (excluding repeated preloaded-context tokens) by up to 46.37%, while maintaining comparable task performance. These findings provide a useful reference for future research in this direction.

1 INTRODUCTION

As the number of reusable Skills continues to grow, Skill Retrieval is becoming an increasingly important component of how Agents access and use external skills (Li et al., 2026a; Zheng et al., 2026; Cho et al., 2026). Existing research primarily focuses on retrieving Skills that are functionally relevant to the current task and can help an Agent complete it, leading to a range of methods centered on relevance matching and effective recall (Li et al., 2026a; Zheng et al., 2026; Goulart et al., 2026).

However, this relevance-oriented retrieval objective mainly answers the question of "which Skills can support task completion." When multiple candidate Skills are functionally viable, it is insufficient to further determine "which Skills can better support efficient downstream Agent execution." As illustrated in Figure 1, multiple functionally relevant Skills for the same task may correspond to different technical routes and lead to substantially different downstream execution costs. Skill Retrieval could therefore not only identify usable Skills, but also make choices among multiple functionally viable candidates so that the selected Skills better support efficient Agent execution. In this work, we consider downstream execution cost as an additional criterion for such selection and define this problem as **Cost-Aware Skill Retrieval**.

However, few existing Skill Retrieval benchmarks support systematic study of this problem. Existing benchmarks are typically constructed around mappings between tasks and relevant Skills, and primarily evaluate the recall, matching, or composition capabilities of retrieval methods (Cho et al., 2026; Wang et al., 2026; Ding et al., 2026). Such designs can evaluate a Retriever's ability to recall functionally effective Skills, but provide limited support for assessing its ability to distinguish among candidate Skills associated with different execution costs. Evaluating Cost-Aware Skill Retrieval re-

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.

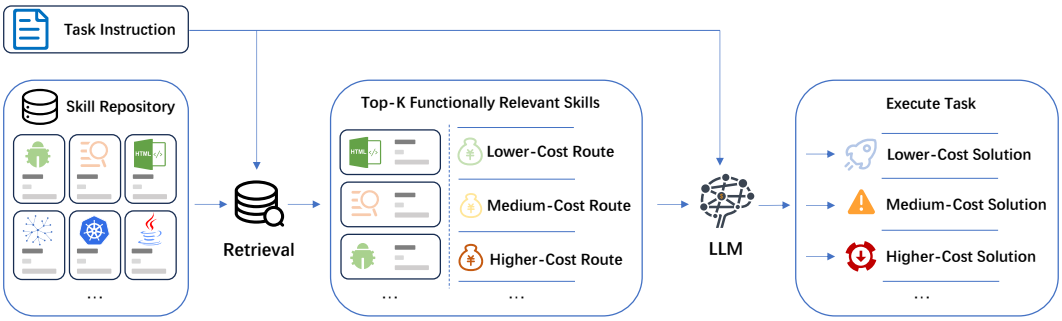

Figure 1: Limitation of Skill Retrieval based solely on functional relevance. Functionally relevant Skills may guide the Agent toward different technical routes and consequently incur substantially different downstream execution costs, while existing methods have limited ability to prioritize more execution-efficient candidates accordingly.

quires constructing, for the same task, a set of Skills that are functionally valid but differ in execution cost. Existing benchmarks generally lack such systematic, execution-cost-oriented designs.

To support systematic evaluation of Cost-Aware Skill Retrieval, we construct the **Cost-Aware Skill Retrieval Benchmark**, which is built from reusable Skills derived from functionally valid reference routes with different execution costs. Controlled Agent-execution experiments further provide empirical support for the intended benchmark construction.

Based on this benchmark, we further evaluate existing Skill Retrieval methods. The results show that strong recall of task-supporting Skills does not necessarily translate into a consistent preference for more execution-efficient Skills in existing retrieval methods. Based on these observations, we further propose a **Cost-Aware Reference Method**. Based on the initial retrieval results, the method performs two stages of candidate technical-route construction and expected execution-cost comparison to further screen and select functionally relevant Skills. Experimental results indicate that the method can favor lower-cost Skills and translate this preference into actual downstream savings, with Agent execution time reduced by up to 51.55% and context-adjusted token consumption reduced by up to 46.37%, while maintaining comparable task performance.

The main contributions of this work are summarized as follows:

- We point out that Skill Retrieval could go beyond identifying functionally relevant Skills and further consider how different viable candidates affect downstream Agent execution costs. Based on this observation, we introduce **Cost-Aware Skill Retrieval** as a research problem.
- We construct and validate the **Cost-Aware Skill Retrieval Benchmark**, providing an evaluation basis for systematically studying the execution costs associated with different retrieval choices. Experiments on this benchmark show that existing retrieval methods achieve strong recall of task-supporting Skills yet lack a consistent preference for more execution-efficient candidates.
- We further propose a **Cost-Aware Reference Method** and present preliminary evidence that it can favor lower-cost Skills and translate this preference into measurable downstream execution savings while maintaining comparable task performance.

2 Related Work

2.1 Skill Retrieval for LLM Agents

As reusable Skill repositories continue to grow, Skill Retrieval has become an important component of Skill-augmented Agents. Early work largely follows classical information retrieval methods, including sparse lexical retrieval with BM25 (Robertson & Zaragoza, 2009) and dense retrieval based on semantic representations (Karpukhin et al., 2020). Recent studies further tailor retrieval

to Skill structure and usage, with SkillFlow (Li et al., 2026a), SkillRouter (Zheng et al., 2026), and Field-Aware Skill Retrieval (Goulart et al., 2026) improving Skill matching through multi-stage retrieval, reranking, or structured field modeling.

Subsequent work has further extended Skill Retrieval to structural and compositional relationships. Graph-of-Skills (Liu et al., 2026a) and Group of Skills (Zeng et al., 2026) model dependencies among Skills and their execution contexts, while Skill Is Not Document (Wang et al., 2026) and Compositional Skill Routing (Gao, 2026) further consider compatibility, task decomposition, and composition in multi-Skill settings. Overall, existing Skill Retrieval research remains centered on functional relevance, with limited consideration of downstream execution cost.

2.2 Benchmarks for Skill Retrieval

Skill Retrieval benchmarks have expanded from evaluating downstream Skill utility to large-scale retrieval settings. SkillsBench (Li et al., 2026b) evaluates the contribution of Skills to downstream task performance, SkillRet (Cho et al., 2026) formulates Skill Retrieval as a standalone task and evaluates retrieval and ranking over large candidate pools, and SRA-Bench (Su et al., 2026) further introduces large numbers of distractor Skills while jointly evaluating Skill retrieval, incorporation, and loading.

Recent benchmarks further consider multi-Skill compatibility, compositional tasks, and discrimination among functionally similar Skills. R3-Skill (Wang et al., 2026) studies query-conditioned compatibility, CompSkillBench (Gao, 2026) focuses on task decomposition and multi-Skill routing, and SameCapRisk-Bench (Ding et al., 2026) evaluates discrimination among Skills with similar capabilities but different applicability conditions. Despite this broader scope, existing benchmarks remain primarily centered on functional retrieval utility and lack dedicated designs for evaluating differences in downstream execution cost induced by different retrieval choices.

2.3 Agent Efficiency and Cost-Aware Decision Making

Beyond task success, execution efficiency has increasingly become an important objective for evaluating and optimizing LLM Agents. Existing work improves efficiency at different levels, including planning mechanisms, workflow design, and system execution, with the goal of reducing execution time, Token consumption, and other resource costs. Interactive Speculative Planning (Hua et al., 2025) and Dynamic Speculative Agent Planning (Guan et al., 2026) primarily focus on planning efficiency, while PatchPilot (Li et al., 2025), SWE-Effi (Fan et al., 2025), and SearchAgent-X (Yang et al., 2026) improve efficiency through workflow and execution-level optimization.

Cost information has also been incorporated into upstream decision making. In model routing, FrugalGPT (Chen et al., 2024), RouteLLM (Ong et al., 2025), and OmniRouter (Mei et al., 2025) jointly consider model capability and inference cost. In tool-use settings, Budget-Aware Tool-Use (Liu et al., 2026b), Scores Are Not Decisions (Feng et al., 2026), and Latency-Quality Routing (Chu et al., 2026) incorporate budget, tool cost, or execution latency into selection decisions. These studies show that execution cost can serve not only as a post-hoc evaluation metric, but also as a signal for upstream decision making.

However, existing cost-aware research has mainly focused on model selection, tool use, and execution strategies, while the role of cost in Skill Retrieval remains underexplored. Unlike direct model or tool selection, Skills affect downstream execution cost indirectly by shaping an Agent's technical routes and execution processes, motivating the incorporation of downstream cost into Skill evaluation and selection.

3 Cost-Aware Skill Retrieval Benchmark

3.1 Problem Formulation

Let $t \in \mathcal{T}$ denote a task and $\mathcal{S}$ a repository of reusable Skills. Given t and $\mathcal{S}$, a Skill Retriever $\mathcal{R}$ retrieves a set of Skills that are functionally relevant to the current task:

$$\mathcal{R}(t, \mathcal{S}) \to \mathcal{S}_t, \qquad \mathcal{S}_t \subseteq \mathcal{S}. \tag{1}$$

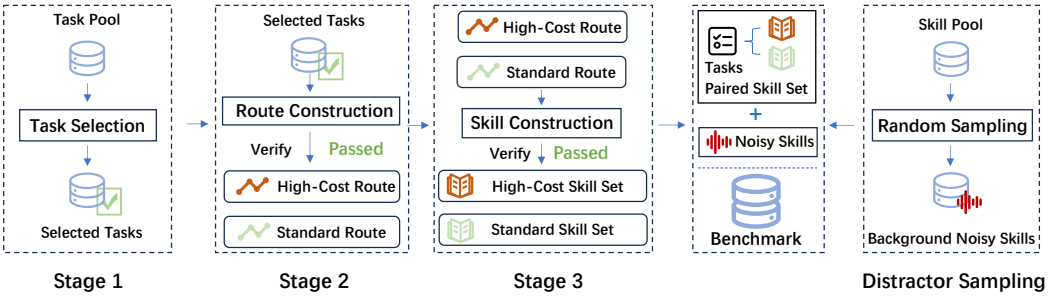

Figure 2: Overall construction pipeline of the Cost-Aware Skill Retrieval Benchmark. We first select tasks suitable for execution-cost comparison and construct and validate a Standard Route and a High-Cost Route for each task. We then derive and review the corresponding Standard and High-Cost Skill Sets from these reference routes. Finally, we add background Noisy Skills randomly sampled from an external Skill Pool to form the final benchmark.

Existing Skill Retrieval methods primarily rely on functional relevance, focusing on whether the retrieved Skills match the task requirements and provide some guidance for Agent execution.

Given an Agent $\mathcal{A}$, the retrieved Skills and the task instruction jointly guide the subsequent execution:

$$\mathcal{A}(t, \mathcal{S}_t) \rightarrow (y_t, \mathbf{c}_t)\,, \tag{2}$$

where y_t denotes the task outcome and $\mathbf{c}_t$ denotes the downstream execution-cost vector, including execution time, memory consumption, Token consumption, and other measurable costs. For the same task, different Skill sets may provide comparable functional utility while corresponding to different technical routes, thereby leading to different execution processes and downstream costs.

Based on this observation, we define Cost-Aware Skill Retrieval as retrieving from functionally relevant candidate Skills while further considering their expected downstream execution costs and prioritizing Skill sets with lower expected costs. Let $\mathcal{C}(t, \mathcal{S})$ denote a collection of functionally relevant candidate Skill sets for task t, $\widehat{\mathbf{c}}_{\text{exec}}(\mathcal{A}, t, \mathcal{S}')$ the expected downstream execution-cost vector of a candidate set $\mathcal{S}'$, and $g(\cdot)$ an application-dependent aggregation function over the considered cost dimensions. The objective is formulated as:

$$\mathcal{S}_t^* = \arg \min_{\mathcal{S}' \in \mathcal{C}(t,\mathcal{S})} g(\widehat{\mathbf{c}}_{\text{exec}}\left(\mathcal{A}, t, \mathcal{S}'\right))\,. \tag{3}$$

Here, functional relevance constrains the valid candidate space, while expected execution cost provides an additional criterion for distinguishing among functionally matched candidates. In our benchmark, cost separation is established primarily by execution time, with memory and Token consumption reported as complementary dimensions; actual downstream costs are obtained through Agent execution.

3.2 Benchmark Construction and Validation

This section presents the overall construction process of the Cost-Aware Skill Retrieval Benchmark. Our primary goal is to construct, for each task, paired Skill sets that provide comparable functional guidance while corresponding to different downstream execution costs, and to further establish a unified retrieval environment with a large number of distractor Skills. Additional details on task selection, reference route construction and measurement, Skill construction, and the experimental environment are provided in Appendices B.1, B.2, B.3, and D, respectively.

As shown in Figure 2, the benchmark construction consists of three main stages. First, we select tasks with diverse solution spaces that are suitable for execution-cost comparison. Second, we construct and validate solutions based on different technical strategies, forming the corresponding Standard Routes and High-Cost Routes. Third, we derive the corresponding Standard Skill Sets and High-Cost Skill Sets from the two types of reference routes. Finally, we incorporate background Noisy Skills randomly sampled from an external Skill Pool to form the final benchmark.

Stage 1: Task Selection. The candidate tasks in our benchmark are drawn from Terminal-Bench 2[1] and Skill-Usage[2]. Following the evaluation objective of Cost-Aware Skill Retrieval, we select tasks according to two main principles: (1) **Route Diversity**: tasks should provide sufficient solution flexibility to support multiple reasonable routes based on different core technical approaches; and (2) **Cost Measurability**: tasks should have sufficient task scale, data scale, or computational complexity such that execution-cost differences induced by different technical routes can be reliably measured and distinguished.

Stage 2: Reference Route Construction. For each selected task, we first treat the technical route underlying its original reference solution as the Standard Route, and then design an alternative High-Cost Route based on a substantively different technical strategy. Solution Design follows three main principles: (1) **Task Faithfulness**: each solution should constitute a complete and practically executable solution to the task and should pass the original task tests; (2) **Strategy Distinctness**: the two solutions should differ substantially in their core technical strategies, rather than only in parameter settings or local implementation details; and (3) **Cost Authenticity**: potential cost differences between the two solutions should arise from genuine technical choices and execution procedures, rather than from artificially introduced computation or waiting overhead.

Table 1: Paired execution-cost statistics for Standard and High-Cost Routes. ST and HC denote the Standard Route and High-Cost Route, respectively; the execution cost of each task-route pair is summarized by the median over three post-warm-up runs.

Metric	Median Overhead	IQR (Q1–Q3)	Geo. Mean Ratio	HC $>$ ST
Execution Time	$+240.6\%$	$+92.6\%$ to $+722.5\%$	$4.22\times$	$35/35$
Peak Memory	$+16.6\%$	-30.6% to $+126.3\%$	$1.27\times$	$21/35$

On the basis of the constructed route pairs, we further validate the Standard and High-Cost Routes through the original task tests and repeated execution under a unified measurement protocol. As shown in Table 1, the High-Cost Route incurs higher execution time on all 35 tasks, with a median overhead (*Median Overhead*) of 240.6% and a geometric mean HC/ST ratio (*Geo. Mean Ratio*) of $4.22\times$, establishing a clear execution-cost separation between the two route types. Peak memory also shows an overall increase, with a median overhead (*Median Overhead*) of 16.6% and higher usage for the High-Cost Route on 21 of the 35 tasks.

Stage 3: Skill Construction. After obtaining the Standard Routes and High-Cost Routes, we extract transferable technical knowledge, operational procedures, and process guidance from the two types of reference routes to construct the corresponding Standard Skill Sets and High-Cost Skill Sets. Depending on the complexity of a reference route, each route may correspond to one or multiple Skills with relatively independent functions that can be composed when needed. Skill Construction follows three main principles: (1) **Skill Reusability**: each Skill should abstract and preserve technical knowledge and operational procedures that can be transferred to similar tasks, rather than being specific to the source task; (2) **Instance Independence**: Skills should not contain final answers, instance-specific parameters, or other information that directly reveals the solution to the source task; and (3) **Route Fidelity**: Skills constructed from different reference routes should preserve the core technical choices and execution logic of their respective routes, so that the guidance they provide remains aligned with the corresponding reference route.

3.3 Benchmark Overview

The resulting Cost-Aware Skill Retrieval Benchmark contains 35 tasks, together with 87 Standard Skills and 85 High-Cost Skills constructed from the corresponding Standard Routes and High-Cost Routes. To approximate retrieval in a large-scale Skill repository, we randomly sample 2,000 Skills from the publicly available dataset `Shiyu-Lab/Skill-Usage`[3] as background distractors. After removing duplicate Skills, we further conduct manual relevance screening to exclude Skills that are

[1] `https://github.com/harbor-framework/terminal-bench-2`
[2] `https://github.com/UCSB-NLP-Chang/Skill-Usage`
[3] `https://huggingface.co/datasets/Shiyu-Lab/Skill-Usage`

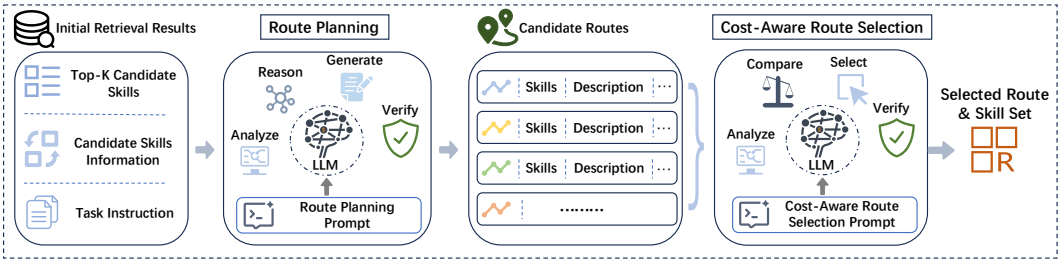

Figure 3: Overall workflow of the Cost-Aware Reference Method. Given a Task Instruction and the Top-K candidate Skills returned by an initial Retriever, the LLM first analyzes the task and candidate Skill information under the Route Planning Prompt, constructs multiple candidate technical routes, and generates the corresponding route descriptions and required Skills; it then compares the expected execution costs of the candidate routes under the Cost-Aware Route Selection Prompt, selects the route with the lowest estimated execution cost, and returns the corresponding Skill set.

functionally relevant to the benchmark tasks. This process results in 1,991 Noisy Skills, yielding a retrieval pool of 2,163 candidate Skills in total.

For each task, the corresponding Standard Skill Set and High-Cost Skill Set are both treated as functionally relevant target Skill sets, while representing technical routes with different downstream execution costs. The Noisy Skills serve as background candidates that constitute the large-scale retrieval environment. This setting enables subsequent experiments to evaluate, within a unified candidate pool, both the ability of a Retriever to identify functionally relevant Skills and its ability to distinguish between relevant Skills associated with different execution costs. The complete benchmark data are available in the repository Benchmark repository.

4 Cost-Aware Reference Method

To provide an initial validation of the feasibility of explicitly incorporating downstream execution cost into Skill retrieval, we develop a lightweight LLM-based Cost-Aware Reference Method. The method operates on candidate Skills returned by existing functional-relevance-based Retrievers (e.g., BM25 and dense retrieval), and further introduces explicit analysis of potential technical routes and their execution costs to improve the selection of lower-cost Skills.

As shown in Figure 3, the method first constructs feasible technical routes based on the task and candidate Skills, and then compares the expected execution costs of different routes to determine the final Skill selection. This section presents the overall design and core workflow of the Reference Method, while more detailed implementation settings and supplementary statistics are provided in Appendix C. The complete prompts and implementation code are available in the repository Retrieval and Selection repository.

4.1 Route Planning

First, route planning is performed based on the Task Instruction and the Top-K candidate Skills returned by the initial Retriever. To limit additional Token overhead, we use only the `skill_name` and `description` fields from `SKILL.md` as the candidate Skill information. Guided by the Route Construction Prompt, the LLM jointly considers the task requirements and candidate Skill information to construct several feasible and technically distinct end-to-end execution routes. For each route, the LLM mainly generates a corresponding technical route description and identifies the Skills actually required for its implementation.

4.2 Cost-Aware Route Selection

Subsequently, cost-aware route selection is performed based on the Task Instruction and the candidate route descriptions generated above. Guided by the Route Selection Prompt, the LLM compares the expected execution costs of the candidate routes according to their concrete execution processes.

The evaluation jointly considers execution time, memory usage, and implementation complexity, following this order when resolving trade-offs across cost dimensions. The LLM then selects the route with the lowest expected overall execution cost and returns the corresponding Skill set.

5 EXPERIMENTAL

5.1 EXPERIMENTAL SETUP

Retrieval Setting. All retrieval experiments are conducted on the unified Skill repository constructed in our benchmark, which contains 87 Standard Skills, 85 High-Cost Skills, and 1,991 Noisy Skills, for a total of 2,163 candidate Skills. For each task, the Retriever ranks the candidate Skills based on the Task Instruction and its corresponding input representation and returns the Top-K results. For the Cost-Aware Reference Method, we use the Top-10 Skills returned by the initial Retriever as the candidate set for subsequent route planning and cost-aware selection.

Retrieval Baselines. We evaluate three representative Skill retrieval methods, covering sparse lexical retrieval, dense semantic retrieval, and Skill-specific retrieval. **BM25** (Robertson & Zaragoza, 2009) constructs the query from the task ID, domain, instruction, and tags, and indexes Skills using the Skill name and frontmatter description. **Qwen Dense Retrieval** (Zhang et al., 2025) uses Qwen3-Embedding-4B with the same Task and Skill fields as BM25, ranking candidate Skills by semantic similarity. **SkillRouter** (Zheng et al., 2026) adopts a two-stage retrieve-and-rerank framework, first retrieving candidates using the Task Instruction and full Skill-document representations, and then reranking them using the Task Instruction, Skill name, frontmatter description, and complete `SKILL.md` content.

Evaluation Metrics. For existing Skill Retrievers, we report **Recall@**5 and **Recall@**10 separately for Standard and High-Cost Skills. **Recall@**K measures target-Skill coverage in the Top-K retrieval results and is macro-averaged across tasks.

For agent-level evaluation, we report **Pass Rate**, **Agent Execution Time**, **Agent Execution Calls**, **Full Agent Tokens**, and **Context-Adjusted Agent Tokens**. These metrics capture task completion and execution cost, with Context-Adjusted Agent Tokens reducing the direct contribution of initial Skill-context differences.

For the Cost-Aware Reference Method, we further report **Final Standard/High-Cost Recall** and **Standard/High-Cost Remain Rate**, measuring target-Skill coverage after selection and the retention of initially retrieved Skills, respectively. Average Token consumption and execution time are also reported as selection overhead.

Unless otherwise specified, further experimental and evaluation details are provided in Appendix D.

5.2 BENCHMARK VALIDATION

To validate whether the constructed paired Skill sets satisfy the core requirements of the Cost-Aware Skill Retrieval Benchmark, we conduct a controlled agent-level evaluation. For each task, we bypass the retrieval stage and directly provide the Agent with the corresponding Standard Skill Set or High-Cost Skill Set, while using a No Skill setting as a reference without external Skill guidance. Experiments are conducted with GPT-5.6-Luna and DeepSeek-V4-Flash, with three independent runs for each task-setting pair. We use Pass Rate to measure task-completion performance, and treat execution time as the primary measure of downstream Agent cost, with Full Token consumption and the number of model calls (Calls) as complementary indicators.

As shown in Table 2, both Standard Skill Sets and High-Cost Skill Sets improve the Pass Rate of the two Agents compared with the No Skill setting, indicating that both types of Skills provide effective guidance for task execution. Meanwhile, under the same Agent, their Pass Rates remain relatively close, suggesting similar task-level effectiveness. This pattern is observed for both Agents, indicating a degree of cross-Agent stability.

Despite their similar task-level performance, Standard Skill Sets and High-Cost Skill Sets exhibit clear differences in downstream execution cost. For both Agents, High-Cost Skill Sets lead to substantially longer execution time, while also incurring higher Full Token consumption and more

Table 2: Agent-level validation of the constructed Standard and High-Cost Skill Sets, where Exec. Time denotes agent execution time excluding verifier overhead. Results are averaged over three runs per task.

Agent	Setting	Pass ↑	Exec. Time (s) ↓	Full Tokens ↓	Calls ↓
GPT-5.6-Luna	No Skill	45.71%	190.71	387,833	20.88
	Standard Skills Set	74.29%	130.13	293,439	15.74
	High-Cost Skills Set	70.48%	342.12	702,840	29.90
DeepSeek-V4-Flash	No Skill	36.19%	472.19	1,490,775	42.69
	Standard Skills Set	73.33%	224.35	1,049,549	32.51
	High-Cost Skills Set	72.38%	546.03	1,177,524	37.25

model calls overall. These results show that Skill sets with similar task-level effectiveness can still induce substantially different execution costs by corresponding to different technical routes. This pattern is observed across both evaluated Agents, supporting the use of the constructed Standard and High-Cost Skill Sets for evaluating Cost-Aware Skill Retrieval.

To further examine whether these cost differences are primarily driven by failed runs or timeouts, we conduct a paired analysis restricted to runs in which both the Standard Skill and High-Cost Skill conditions successfully complete the task. As reported in Appendix E, High-Cost Skills continue to incur longer execution time and higher Token consumption under this restricted comparison.

5.3 Evaluation of Existing Skill Retrievers

To evaluate whether existing Skill Retrievers can distinguish between functionally relevant Skills associated with different downstream execution costs, we compare three representative retrieval methods on the constructed benchmark and report Recall@5 and Recall@10 for Standard Skills and High-Cost Skills, examining whether they exhibit a stable preference for lower-cost Skills.

Table 3: Recall of Standard and High-Cost Skills for existing Skill retrieval methods.

Method	St R5	HC R5	St R10	HC R10
BM25	0.7762	0.8071	0.8833	0.8690
Qwen Dense	0.7619	0.8500	0.9190	0.9286
SkillRouter	0.8667	0.8310	0.9214	0.9048

As shown in Table 3, all three existing Skill Retrieval methods achieve relatively strong recall of task-relevant Skills, but do not exhibit a stable preference between Standard Skills and High-Cost Skills. At $K = 10$, the Recall values of the two Skill types are largely comparable. At $K = 5$, the High-Cost Recall of BM25 and Qwen Dense is even higher than their Standard Recall, while SkillRouter shows only a limited advantage for Standard Skills.

These results suggest that, for the evaluated functional-relevance-based Skill Retrievers, strong task-relevance retrieval does not necessarily provide further discrimination with respect to downstream execution cost. When multiple candidate Skills are functionally relevant to the task, existing Retrievers do not consistently favor those associated with lower execution costs, and High-Cost Skills may still be retrieved at rates comparable to, or even higher than, those of Standard Skills. Functional relevance alone is therefore insufficient to consistently prioritize Skill choices that support more efficient downstream Agent execution.

5.4 Evaluation of Cost-Aware Reference Method

To evaluate whether the proposed Cost-Aware Reference Method can establish a stable preference for lower-cost Skill selection across different initial Retrievers and LLM backbones, we further evaluate its final selection results and additional overhead.

As shown in Table 4, the Cost-Aware Reference Method exhibits a consistent preference for lower-cost Skill selection across different initial Retrievers, with this tendency being more pronounced under GPT-5.6-Luna. Across all initial Retrievers, Standard Skills achieve higher Final Recall and Remain Rate than their High-Cost counterparts. The consistently higher Standard Remain Rate indicates that the method does not uniformly reduce the candidate Skill set, but instead preferentially retains Standard Skills while reducing High-Cost Skills.

Table 4: Evaluation of the Cost-Aware Reference Method with different backbone models.

Retriever	Backbone	Final Recall		Remain Rate		Overhead	
		ST ↑	HC ↓	ST ↑	HC ↓	Tokens (K) ↓	Time (s) ↓
BM25	GPT-5-nano	0.6206	0.4825	0.6817	0.5662	11.76	35.20
	GPT-5.6-Luna	0.5556	0.3095	0.6119	0.3587	7.48	36.17
Qwen Dense	GPT-5-nano	0.5698	0.5071	0.6127	0.5359	11.37	48.17
	GPT-5.6-Luna	0.5508	0.3087	0.5931	0.3284	8.33	52.61
SkillRouter	GPT-5-nano	0.5960	0.4548	0.6230	0.4984	11.55	53.89
	GPT-5.6-Luna	0.5897	0.2381	0.6437	0.2500	9.13	38.29

In contrast, under GPT-5-nano, the gaps between Standard and High-Cost Skills in both Final Recall and Remain Rate are generally smaller, indicating more limited cost-aware differentiation and suggesting that the effectiveness of the Reference Method is sensitive to backbone choice. In addition, GPT-5.6-Luna incurs substantially lower average Token consumption than GPT-5-nano, which may reflect differences in reasoning efficiency during route planning and cost comparison across backbone models.

Overall, these results provide preliminary evidence that explicitly analyzing candidate technical routes and their expected execution costs can shift the final Skill selection toward lower-cost candidates, with Standard Skills being retained at consistently higher rates than High-Cost Skills. At the same time, the current method remains dependent on backbone capability and introduces additional inference overhead.

To further examine whether the above selection-level preference translates into practical execution benefits, we provide additional Agent-execution and end-to-end evaluations in Appendix F. Compared with directly using the Top-10 Skills returned by the initial Retriever, cost-aware selection reduces Agent execution time by up to 51.55% and context-adjusted token consumption by up to 46.37%. Some configurations also achieve lower end-to-end token consumption and execution time after accounting for retrieval and selection overhead.

6 Discussion

Limitations and Future Directions. Our current study focuses on downstream execution-cost differences induced by different technical routes, and the resulting benchmark remains limited in task scale, task diversity, and Skill sources. The current cost characterization primarily covers execution time, peak memory usage, and token consumption, whereas practical Agent systems may involve additional dimensions such as tool-use expenses and external service latency. In addition, the Cost-Aware Reference Method is intended as an initial validation of explicit cost-aware Skill selection; its effectiveness can vary across backbone models and it introduces additional inference overhead. Future work can broaden the range of tasks and Skills, incorporate richer cost dimensions, and explore more efficient cost estimation and end-to-end cost-aware retrieval methods.

Conclusion. In this work, we introduce Cost-Aware Skill Retrieval, which incorporates downstream execution cost into Skill evaluation and selection, and construct a corresponding benchmark. Experiments show that functionally effective Skills can still lead to substantially different downstream execution costs, while existing Skill Retrievers can effectively recall relevant Skills but lack a consistent preference for lower-cost candidates. We further propose a Cost-Aware Reference Method and show that it can preferentially select lower-cost Skills and reduce actual Agent execution costs, demonstrating some practical utility. Finally, we hope this work can provide a useful reference for future research on cost-aware Skill Retrieval.

AI USE STATEMENT

Generative AI tools were used to assist in refining the research design and experimental planning, preparing analysis code, summarizing and interpreting experimental results, conducting literature searches, revising the manuscript, and editing figures and tables.

The High-Cost (HC) routes, their corresponding implementations, and the derived Skills were constructed by the authors. The authors manually reviewed AI-assisted code, analyses, and manuscript content. AI-assisted analyses were cross-checked against the archived experimental records, and literature-based claims in the manuscript were verified against their cited sources.

The authors take full responsibility for the final manuscript and its accompanying materials. LLMs evaluated as planners, selectors, or executors constitute part of the experimental setup and are described separately in the corresponding experimental sections; their use is distinct from the use of generative AI tools for research assistance described above.

REPRODUCIBILITY STATEMENT

The anonymous Benchmark repository provides the Tasks, Skills, and their mappings used in the benchmark, together with the execution and result-verification code, experimental data, and run logs required to reproduce the end-to-end results. The anonymous Retrieval and Selection repository provides the implementation of Skill retrieval and selection, the retrieval evaluation framework, and the corresponding experimental configurations.

Benchmark construction and Cost-Aware Reference Method details are provided in Appendices B and C, with complete prompts and reproduction materials released in the anonymous repositories. Experimental environments, measurement protocols, metrics, Retriever implementations, and evaluation settings are provided in Appendix D.

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*, 2024.

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. SkillRet: A large-scale benchmark for skill retrieval in LLM agents. *arXiv preprint arXiv:2605.05726*, 2026.

Kexin Chu, Dawei Xiang, and Wei Zhang. Latency-quality routing for functionally equivalent tools in LLM agents. *arXiv preprint arXiv:2605.14241*, 2026.

Jiandong Ding, Honglei Ji, Ming Liu, and Tao Duan. Right family, wrong skill: Evaluating risk exposure in agent skill retrieval. *arXiv preprint arXiv:2606.10388*, 2026.

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, et al. SWE-Effi: Re-evaluating software AI agent system effectiveness under resource constraints. *arXiv preprint arXiv:2509.09853*, 2025.

Yicheng Feng, Yan Zhang, Yan Cheng, and Wei Qi. Scores are not decisions: Cost-aware stopping for tool acquisition in LLM agents. *arXiv preprint arXiv:2607.27083*, 2026.

Xueping Gao. Compositional skill routing for LLM agents: Decompose, retrieve, and compose. *arXiv preprint arXiv:2606.18051*, 2026.

Paimon Goulart, Liang Wu, Kelly Wan, Evangelos E. Papalexakis, and Liangjie Hong. Field-aware agent skill retrieval. *arXiv preprint arXiv:2608.02880*, 2026.

Yilin Guan, Qingfeng Lan, Fei Sun, et al. Dynamic speculative agent planning. In *International Conference on Learning Representations*, 2026.

Wenyue Hua, Mengting Wan, Jagannath Shashank Subramanya Sai Vadrevu, Ryan Nadel, Yongfeng Zhang, and Chi Wang. Interactive speculative planning: Enhance agent efficiency through co-design of system and user interface. In *International Conference on Learning Representations*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, et al. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, 2020. Association for Computational Linguistics.

Fangzhou Li, Pagkratios Tagkopoulos, and Ilias Tagkopoulos. SkillFlow: Scalable and efficient agent skill retrieval system. *arXiv preprint arXiv:2504.06188*, 2026a.

Hongwei Li, Yuheng Tang, Shiqi Wang, and Wenbo Guo. PatchPilot: A cost-efficient software engineering agent with early attempts on formal verification. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 35922–35941. PMLR, 2025.

Xiangyi Li, Yimin Liu, Wenbo Chen, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Dawei Liu, Zongxia Li, Hongyang Du, et al. Graph-of-Skills: Dependency-aware structural retrieval for massive agent skills. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026a.

Tengxiao Liu, Zifeng Wang, Jin Miao, et al. Budget-aware tool-use enables effective agent scaling. In *Third Conference on Language Modeling*, 2026b.

Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. OmniRouter: Budget and performance controllable multi-LLM routing. *ACM SIGKDD Explorations Newsletter*, 27 (2):107–116, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, et al. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025.

Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Weihang Su, Jianming Long, Qingyao Ai, et al. Skill retrieval augmentation for agentic AI. *arXiv preprint arXiv:2604.24594*, 2026.

Zifei Wang, Wei Wen, Qiang Ji, Keyu Chen, Ruizhi Qiao, and Xing Sun. Skill is not document: Query-conditioned compatibility for LLM agent skill routing. *arXiv preprint arXiv:2606.03565*, 2026.

Tiannuo Yang, Zebin Yao, Bowen Jin, et al. Demystifying and enhancing the efficiency of large language model based search agents. In *International Conference on Learning Representations*, 2026.

Kun Zeng, Yu Huo, Siyu Zhang, et al. Group of skills: Group-structured skill retrieval for agent skill libraries. *arXiv preprint arXiv:2605.06978*, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, et al. SkillRouter: Skill routing for LLM agents at scale. *arXiv preprint arXiv:2603.22455*, 2026.

A Benchmark Comparison

Table 5 compares representative Agent Skill benchmarks in terms of benchmark design and supported evaluation capabilities. Existing benchmarks primarily focus on functional relevance, applicability, or multi-Skill composition, but generally do not explicitly construct multiple functionally viable Skill alternatives for the same task with different downstream execution costs. Our benchmark supports Multi-Skill settings, Skill Retrieval, and Skill Composition, while further introducing functionally viable alternatives to enable the evaluation of Cost-Aware Skill Retrieval. Here, # Tasks denotes the number of evaluation tasks or queries, and # Skills denotes the reported Skill candidate-pool size; N/R indicates that the corresponding number is not reported.

Table 5: Comparison of representative Agent Skill benchmarks in terms of benchmark design and supported evaluation capabilities.

Benchmark	Benchmark Design				Evaluation Capabilities		
	# Tasks	# Skills	Multi-Skill	Viable Skill Alternatives	Skill Retrieval	Skill Composition	Cost-Aware Retrieval
SkillsBench (Li et al., 2026b)	87	N/R	✓	×	×	✓	×
SkillRet (Cho et al., 2026)	4,997	6,660	✓	×	✓	✓	×
SRA-Bench (Su et al., 2026)	5,400	26,262	✓	×	✓	✓	×
R3-Skill (Wang et al., 2026)	5,696	2,050	✓	×	✓	✓	×
CompSkillBench (Gao, 2026)	300	2,209	✓	×	✓	✓	×
SameCapRisk-Bench (Ding et al., 2026)	1,686	8,048	×	×	✓	×	×
Ours	**35**	**2,163**	✓	✓	✓	✓	✓

B Benchmark Construction Details

B.1 Task Selection Details

The initial candidate pool consists of 89 tasks from Terminal-Bench 2 and 87 tasks from Skill-Usage. We first perform multiple rounds of screening based on task descriptions and the potential solution space, retaining tasks that admit multiple reasonable technical routes while excluding those with highly constrained solution paths that are unlikely to exhibit substantive route-level differences.

We then benchmark the reference solution provided with each task to further identify tasks with sufficient execution scale and relatively stable resource consumption. For each task, we first perform one warm-up run, followed by three valid measurement runs. Runs affected by evident network fluctuations or other external disturbances are excluded and repeated. We require the reference solution to take more than 10 seconds to execute, ensuring sufficient execution scale for subsequent cost comparison. All measurements follow the execution environment and measurement protocol described in Appendix D.

B.2 Reference Route Construction

Route Design and Review. For each candidate task retained after the initial screening, we treat the technical route underlying the original reference solution as the Standard Route and design an alternative High-Cost Route based on a different core technical strategy. The High-Cost Route is required to differ substantively from the Standard Route in both its core technical choices and execution procedure, rather than relying only on parameter adjustments or local implementation changes.

Once a candidate High-Cost Route is developed, two researchers independently cross-review its technical strategy and execution procedure. The review focuses on whether the route differs meaningfully from the Standard Route at the level of core technical choices, and whether its expected additional execution cost arises from those choices themselves rather than from artificially introduced overhead, such as explicit waiting, meaningless loops, or deliberately redundant computation. Candidate routes that do not satisfy these criteria are redesigned and reviewed again. When the two reviewers disagree, the route is further revised and re-evaluated until agreement is reached. Candidate tasks for which a valid route pair cannot be established are excluded from subsequent validation.

Route Validation and Cost Measurement. After the route pair passes the design review, we validate the corresponding Standard and High-Cost solutions using the original task tests to ensure functional correctness. We then evaluate both routes under a unified execution environment and measurement protocol. For each task-route pair, we first perform one warm-up run, followed by three measured runs. The median of the three measured runs is used as the representative execution time and peak memory usage of that route. Standard and High-Cost Routes for the same task are evaluated under the same task and container configuration. Detailed hardware, container settings, and measurement procedures for execution time and peak memory are provided in Appendix D.

To ensure a clear and measurable separation in execution cost, we require the High-Cost Solution to incur at least 50% more execution time than the corresponding Standard Solution, while its peak memory usage must not be more than 50% lower. Only tasks satisfying the requirements of functional correctness, route-level distinction, and execution-cost separation are retained. After this validation process, 35 tasks remain, each associated with a Standard/High-Cost route pair.

To quantify execution-cost differences across tasks, we perform paired comparisons separately for execution time and peak memory usage. Because absolute execution costs can vary substantially across tasks, we summarize the differences using relative overhead and cost ratios. For task t and cost metric m, let $C_{t,\mathrm{ST}}^{(m)}$ and $C_{t,\mathrm{HC}}^{(m)}$ denote the median costs of the Standard Route and High-Cost Route, respectively, over the three measured runs. We define the task-level relative overhead and cost ratio as

$$\mathrm{Overhead}_t^{(m)} = \frac{C_{t,\mathrm{HC}}^{(m)} - C_{t,\mathrm{ST}}^{(m)}}{C_{t,\mathrm{ST}}^{(m)}} \times 100\%, \qquad \mathrm{Ratio}_t^{(m)} = \frac{C_{t,\mathrm{HC}}^{(m)}}{C_{t,\mathrm{ST}}^{(m)}}.$$

We then aggregate these task-level statistics across all retained tasks. In Table 1, *Median Overhead* denotes the median task-level relative overhead, *IQR* denotes the interval from the 25th to the 75th percentile of the task-level overheads, *Geo. Mean Ratio* denotes the geometric mean of the task-level HC/ST cost ratios, and *HC > ST* reports the number of tasks for which the High-Cost Route incurs a higher value on the corresponding cost metric than the Standard Route. These statistics are computed independently for execution time and peak memory usage.

B.3 Skill Construction

After obtaining the validated Standard Route and High-Cost Route, we construct Skills based on the actual execution process of the corresponding Standard and High-Cost Solutions. We first examine the complete execution workflow, identify key subprocesses with relatively independent functions and sufficient generality, and decompose each technical route into one or more Skills accordingly, rather than mechanically following function or code-module boundaries.

We then abstract and organize the information associated with each subprocess. Information specific to the current task instance is not directly encoded into the Skill and is instead obtained from the task context at execution time. In contrast, algorithms, rules, default settings, and key operational procedures determined by the technical route and necessary for execution are preserved. Critical execution logic is described with sufficient detail to allow the Skill to guide the Agent independently without relying on an inaccessible reference implementation. This abstraction removes instance-specific information without redesigning or replacing the underlying technical route.

Once the capability boundary and content of each Skill are determined, we organize them using a consistent structure. The Skill name is based on its primary processing object and core operation, providing a concise representation of its capability. The description summarizes its main functionality, applicable objects or scenarios, and core processing approach. The main body follows a general-to-specific organization: it first presents the overall execution procedure and necessary usage constraints, and then describes individual subprocesses in detail. Key data structures, computational rules, or operations are specified through formulas, templates, examples, or explicit procedural rules, with execution checks and output constraints included where necessary.

After construction, two researchers cross-review the Skills, focusing on whether: (1) task answers, input contents, or other instance-specific information are leaked; (2) the Skill can be reused for tasks with similar requirements; (3) the core technical choices and necessary execution knowledge of the corresponding route are faithfully preserved; and (4) the Skill does not explicitly expose any label or annotation that directly identifies the relative execution cost or preference status of the corresponding technical route. Skills that fail to satisfy these criteria are revised and re-evaluated. If the two reviewers disagree, the Skill is further revised and re-evaluated until both reviewers reach agreement; Skills that still cannot satisfy the review criteria are discarded.

C Cost-Aware Reference Method Details

This section describes the detailed inputs, processing procedure, and structured outputs of the two-stage Cost-Aware Reference Method. The complete prompts used for Route Planning and Cost-Aware Route Selection, together with the corresponding implementation and configuration files, are provided in the repository Retrieval and Selection repository.

C.1 Route Planning

Input Format. The Route Planning stage takes the Task Instruction and the candidate Skills returned by the initial Retriever as input. To limit input overhead, we provide only the name and description of each Skill and assign temporary identifiers to establish explicit associations between candidate routes and Skills. The input is organized as follows:

Route Planning Input Format

```
## Task Instruction
<TASK_INSTRUCTION>

## Top-K Candidate SKILL Knowledge
## Skill reference: S{i}, i = 1, ..., K
Name: <skill_name>
Description: <skill_description>
```

Process and Output. Guided by the Route Planning prompt, the LLM constructs feasible routes with substantively different technical strategies based on the task requirements and candidate Skills, and identifies the Skills required by each route. The output is constrained by a strict JSON Schema to ensure structural consistency and to explicitly associate each route with its corresponding Skill binding:

Route Planning Output Format

```
{
  "task_understanding": {
    "objective": "...", "requirements": [...],
    "required_outputs": [...]
  },
  "routes": [{
    "route_id": "...",
    "summary": "...",
    "core_approach": "...",
    "execution_process": [...],
    "assumptions": [...],
    "completion_confidence": "...",
    "skill_usage": [{"skill_ref": "S{i}", "usage": "..."}]
  }, ...]
}
```

C.2 Cost-Aware Route Selection

Input Format. The Cost-Aware Route Selection stage takes the original Task Instruction and the candidate routes generated in the preceding stage as input. Only the technical description and execution process of each route are retained, while the corresponding Skill bindings are excluded. This design focuses the cost assessment on the technical routes themselves and avoids directly conditioning the selection on the number of Skills or other Skill-level information. The input is organized as follows:

Cost-Aware Route Selection Input Format

```
## Task Instruction
<TASK_INSTRUCTION>
```

```
## Candidate Technical Routes
[
  {"route_id": "...", "summary": "...",
   "core_approach": "...", "execution_process": [...],
   "assumptions": [...]},
  ...
]
```

Process and Output. Guided by the Cost-Aware Route Selection prompt, the LLM comparatively evaluates the candidate routes in terms of execution time, memory usage, and implementation complexity, and selects the route with the lowest expected overall execution cost. The output is likewise constrained by a strict JSON Schema and contains only the selected route and a concise rationale:

Cost-Aware Route Selection Output Format

```
{
  "selected_route_id": "...",
  "selection_reason": "..."
}
```

Finally, we use `selected_route_id` to retrieve the Skills bound to the selected route during Route Planning, which constitute the final Skill selection.

D Experimental Setup

D.1 Hardware and System Environment

All experiments are conducted on the same host machine running Windows 11 25H2 (x64), equipped with an Intel Core i7-14700HX processor (20 cores / 28 logical processors), 64 GB of system memory, and an NVIDIA GeForce RTX 4060 Laptop GPU with 8 GB of VRAM (driver version 566.24). Experiments involving actual task execution are run through Docker Desktop 4.80.0 and Docker Engine 29.6.1 with the WSL2 backend. For task execution, CPU resources are allocated at the physical-core level using only performance cores (P-cores), with each physical P-core assigned to at most one concurrently running container. During execution, nonessential host workloads are minimized, with only necessary system and experiment-related processes retained whenever possible to reduce interference from background activity.

D.2 Direct Task Execution Protocol

Direct task execution is used for task screening and for measuring the execution costs of the Standard and High-Cost Reference Routes. Each task is executed in an isolated Docker container. For a given task, the Standard and High-Cost solutions and all repeated runs use the same container configuration. Each solution is first executed once as a warm-up, followed by three measured runs. The warm-up and subsequent runs are performed within the same container instance to reduce the influence of one-time initialization, dependency loading, or downloading overhead. Sufficient host-side resource headroom is maintained during execution, and network and external execution conditions are kept as consistent as possible across repeated runs.

Execution time is measured as wall-clock time from the start of the corresponding solution script to its termination, excluding verifier execution. Memory usage is sampled over the same interval from the target task container at 20-ms intervals using the cgroup v2 `memory.current` interface, and the maximum observed value is reported as peak memory usage.

D.3 Agent Execution Protocol

Agent-level experiments are likewise conducted in the isolated Docker container associated with each task, using the same task-specific configuration across experimental conditions. Harbor 0.17.1 is used as the execution framework and Codex 1.145.0 as the Agent framework. The temperature is set to 1 for all Agent experiments, the model reasoning effort is set to `medium` and `high` for

GPT-5.6-Luna and Deepseek-V4-Flash, respectively, and the default timeout for an individual model response is 120 seconds. Unless otherwise specified, all remaining Agent and generation parameters follow their default settings.

Agent execution time is defined as the wall-clock time from the start of Agent task execution until the Agent terminates or reaches the task-specific execution time limit, excluding verifier execution. Peak memory usage is measured over the same interval by sampling the target task container's cgroup v2 `memory.current` value at 20-ms intervals and taking the maximum observed value. After Agent execution terminates, the original task verifier is used to determine whether the task has been completed successfully.

To minimize differences caused by whether the Agent discovers or reads Skill files on its own during execution, we use the same **Skill-preloading procedure** in both conditions. Before Agent execution begins, we place the contents of each Skill's root `SKILL.md` file in the initial task context, ensuring that its primary guidance is available from the first model call. Auxiliary files remain in the execution environment and can be accessed by the Agent as needed.

D.4 Retrieval Setup and Implementation Details

D.4.1 Retriever Setup

Runtime Environment. Retrieval experiments are executed in a dedicated Conda environment, `qwen`, on the Windows host, separately from the Docker environments used for task execution. The environment uses Python 3.12.13, PyTorch 2.5.1 with CUDA 11.8, Transformers 5.12.1, and Sentence Transformers 5.6.0. BM25, Qwen Dense Retrieval, and SkillRouter are all evaluated under this common retrieval runtime environment.

Evaluation Settings. All Retrievers operate on the same Skill repository containing 2,163 candidate Skills, with each Skill corresponding to a unique index record. We evaluate existing Retrievers using both Top-5 and Top-10 results. For the Cost-Aware Reference Method, the Top-10 Skills returned by the initial Retriever are used as the candidate set for subsequent selection.

D.4.2 Retrieval Baseline Details

BM25. BM25 implements Skill retrieval using SQLite FTS5. The Task query is constructed by concatenating the task ID, domain, instruction, and tags, while each Skill document consists of the Skill name and frontmatter description, with field weights of 10.0 and 5.0, respectively. Skill documents are indexed offline using SQLite FTS5. At retrieval time, the Task query is processed online, and the built-in BM25 scoring function computes lexical relevance between the query and candidate Skills. Candidates are ranked according to their BM25 scores, and the Top-K Skills are returned. Text uses `unicode61` as the tokenizer. The parameters k_1 and b are set to 1.2 and 0.75, respectively, where k_1 controls the saturation of term-frequency contributions and b controls the strength of document-length normalization.

Qwen Dense Retrieval. Qwen Dense Retrieval uses Qwen3-Embedding-4B as the underlying encoder and performs Skill retrieval based on cosine similarity between Task and Skill representations. The Task query is constructed by concatenating the task ID, domain, instruction, and tags, followed by the retrieval instruction used by the model. Each Skill document is constructed from the Skill name and frontmatter description. Skill embeddings are pre-computed and stored offline. At retrieval time, only the Task query is encoded online and L2-normalized, and its similarity to the stored Skill embeddings is computed by dot product. Since both Task and Skill embeddings are L2-normalized, this score is equivalent to cosine similarity. Candidate Skills are ranked in descending order of similarity, and the Top-K Skills are returned. The embedding dimension is 2,560, the maximum input length is 4,096 tokens, and inference is performed in FP16.

SkillRouter. SkillRouter adopts two stages, Retrieval and Rerank. The Retrieval stage first selects 20 candidate Skills from the complete repository, after which the Rerank stage further evaluates these candidates and returns the final Top-10 Skills.

The Retrieval stage uses SkillRouter-Embedding-0.6B as the encoder. The Task representation contains only the Task Instruction together with the corresponding retrieval instruction, while the Skill representation contains the Skill name, frontmatter description, and Skill body. Skill embeddings are pre-computed offline. At retrieval time, the Task Instruction is encoded online, and normalized Task and Skill embeddings are compared using dot product. Candidate Skills are ranked in descending order of similarity, and the Top-20 candidates are passed to the Rerank stage.

The Rerank stage uses SkillRouter-Reranker-0.6B to score the 20 retrieved candidates. For each candidate, the reranker takes the Task Instruction, Skill name, frontmatter description, and Skill body as input, and computes the logits of the "yes" and "no" tokens. The relevance score is defined as $\text{score} = \text{logit}(\text{yes}) - \text{logit}(\text{no})$, where a higher score indicates stronger predicted relevance to the task. The candidates are then ranked in descending order of this score, and the Top-10 Skills are returned. Ties are resolved according to their order in the Retrieval stage. Both the Retrieval encoder and reranker use a maximum input length of 4,096 tokens and operate in FP16.

D.5 Cost-Aware Reference Method Evaluation Settings

For the Cost-Aware Reference Method, we evaluate each combination of task, initial Retriever, and LLM backbone with three independent runs of Route Planning and Cost-Aware Route Selection to reduce the effect of stochasticity in LLM-based selection.

Final Recall and Remain Rate are computed separately for each independent run. For each task, we first average the metric values over the three runs and then macro-average them across all tasks to obtain the results reported in Table 4. Token consumption and execution time are likewise averaged over the three independent runs when reporting the selection overhead.

Route Planning and Cost-Aware Route Selection use the same generation settings as the Agent experiments: the temperature is set to 1 and the default timeout for an individual model response is 120 seconds. Unless otherwise specified, all other generation parameters follow their default settings.

D.6 Evaluation Metrics

Recall@K. For existing Skill Retrievers, we report Recall@K separately for Standard Skills and High-Cost Skills. Given the target Skill set of type type for task t, denoted as $\mathcal{S}_t^{\text{type}}$, and the Top-$K$ Skills returned by the Retriever, denoted as $\mathcal{R}_K(t)$, we define

$$\text{Recall@}K_t^{\text{type}} = \frac{\left|\mathcal{R}_K(t) \cap \mathcal{S}_t^{\text{type}}\right|}{\left|\mathcal{S}_t^{\text{type}}\right|},$$

where $\text{type} \in \{\text{Standard}, \text{High-Cost}\}$. The final Recall@$K$ is macro-averaged across all tasks.

Final Recall. For the Cost-Aware Reference Method, we further measure the coverage of Standard Skills and High-Cost Skills in the final selected Skill set. Given the final Skill set $\mathcal{S}_t^{\text{final}}$, we define

$$\text{FinalRecall}_t^{\text{type}} = \frac{\left|\mathcal{S}_t^{\text{final}} \cap \mathcal{S}_t^{\text{type}}\right|}{\left|\mathcal{S}_t^{\text{type}}\right|}.$$

The final results are likewise macro-averaged across all tasks.

Remain Rate. Remain Rate measures the proportion of initially retrieved Skills of a given type that are retained after Cost-Aware Route Selection:

$$\text{RemainRate}_t^{\text{type}} = \frac{\left|\mathcal{S}_t^{\text{final}} \cap \mathcal{R}_K(t) \cap \mathcal{S}_t^{\text{type}}\right|}{\left|\mathcal{R}_K(t) \cap \mathcal{S}_t^{\text{type}}\right|}.$$

If no Skill of the corresponding type is retrieved in the initial Top-K results, the Remain Rate for that task and type is treated as `None` and excluded from the macro-average. Standard Remain Rate and High-Cost Remain Rate are used to assess whether the method preferentially retains Standard Skills while filtering out High-Cost Skills.

Pass Rate. For the agent-level evaluation, each task-setting pair is independently executed three times. A run is counted as successful if the Agent completes the task within the prescribed time limit and passes the original task tests. A run is counted as failed if the produced result does not pass the tests or if execution exceeds the task-specific time limit. Pass Rate is computed as the proportion of successful runs over all independent executions.

Execution Time. Execution time is measured from the start of Agent execution until the Agent completes the task or reaches the prescribed time limit. All task images are cached in advance, so image building and downloading time are excluded from the measurement. The execution environment and measurement protocol follow Appendix D.

Agent Execution Calls. For execution r, the number of Agent Execution calls is T_r, i.e., the number of model calls made by the Agent. *Calls* reports the mean of T_r across Agent executions.

Agent Execution Time. Agent Execution Time measures the elapsed time from the start of Agent task execution to the completion of the Agent execution stage, excluding Verifier time. The *Agent Time (s)* column reports its mean across Agent executions.

Full Agent Tokens. For execution r, let T_r denote the number of model calls. We define

$$C_{\text{full}}^{(r)} = \sum_{t=1}^{T_r} \left(I_t^{(r)} + O_t^{(r)} \right),$$

where $I_t^{(r)}$ and $O_t^{(r)}$ denote the input and output tokens of model call t, respectively. This metric counts all input and output tokens across the entire Agent execution and is used to measure the full Agent token consumption. The *Full Tok. (K)* column reports its mean across Agent executions in thousands of tokens.

Context-Adjusted Agent Tokens. Since different supplied Skill sets may induce different initial context lengths, Full Agent Tokens can mix execution-related Token accumulation with repeated static-context overhead. We therefore use the first-call input $I_1^{(r)}$ as an execution-specific baseline:

$$C_{\text{ctx}}^{(r)} = \sum_{t=1}^{T_r} \left(I_t^{(r)} - I_1^{(r)} + O_t^{(r)} \right).$$

This removes the repeated contribution of the initial context while retaining additional input Tokens accumulated during execution and all output Tokens. *Ctx.-adj. Tok. (K)* reports the mean across executions in thousands of Tokens, and is used as a complementary metric to Full Agent Tokens for comparing execution-related Token costs across conditions with different initial context sizes.

E Supplementary Benchmark Validation

To further examine whether the observed execution-cost differences between Standard Skills and High-Cost Skills robustly reflect differences in the downstream execution associated with the two Skill types, we consider two potential confounding factors: (1) whether the observed Token-cost differences are primarily attributable to differences in the number of supplied Skills or the size of the initial Skill context; and (2) whether the observed execution-cost differences are primarily driven by failed runs or timeouts. To address these two questions, we conduct a context-control analysis and a paired-success analysis, respectively.

Controlling for Skill-Set Size and Initial Context Differences. We first compare the Skill-set size and initial Skill-context size under the Standard and High-Cost conditions. Since all supplied Skills are preloaded into the Agent's initial context, differences in the number or length of Skills repeatedly contribute to the input Tokens of subsequent model calls and therefore affect Full Agent Tokens. To reduce the influence of this factor on the comparison, we additionally report Context-Adjusted Agent Tokens. This metric reduces the effect of differences in initial context size on Token-cost comparisons by removing the repeated contribution of the initial input from subsequent model calls, thereby enabling a fairer comparison of execution-related Token consumption across different Skill conditions.

Table 6: Skill-set statistics and Agent token consumption for the Standard (ST) and High-Cost (HC) Skill conditions. Initial Token counts are computed separately for each model using its corresponding tokenization scheme, over the full text of the initially supplied SKILL.md files.

Setting	Avg. # Skills	GPT-5.6-Luna			Deepseek-V4-Flash		
		Initial Tok.	Full Tok.	Ctx.-Adj. Tok.	Initial Tok.	Full Tok.	Ctx.-Adj. Tok.
ST	2.49	9.45	293.44	144.98	11.82	1,049.55	711.05
HC	2.43	9.40	702.84	424.21	11.77	1,177.52	755.87

As shown in Table 6, the Standard and High-Cost conditions have broadly comparable Skill-set sizes and initial Skill-context sizes. More importantly, after controlling for the effect of different initial context sizes on Token-cost comparisons, the High-Cost condition still incurs higher Token consumption for both evaluated Agent models. This indicates that the Token-cost differences observed in the Benchmark Validation cannot be simply attributed to differences in the number or initial context length of the supplied Skills, and are better explained by differences in the downstream execution processes associated with the two Skill conditions.

Controlling for Failed Runs and Timeouts. We next examine whether the observed execution-cost differences are primarily driven by failed runs or timeouts. This analysis directly uses the existing Agent-execution records from the Benchmark Validation experiments, while keeping the Agent models, tasks, Skill settings, execution environment, and number of repeated runs unchanged. The comparison is restricted only to runs in which both the Standard and High-Cost Skill conditions successfully complete the task.

Specifically, Standard and High-Cost runs are paired by Agent model, task, and repetition index. We retain only pairs in which both runs pass the original task verification, terminate normally, and contain no execution errors. Under this criterion, 54 of 105 valid pairs are retained for GPT-5.6-Luna, covering 25 of 35 tasks, while 64 of 105 valid pairs are retained for DeepSeek-V4-Flash, covering 25 of 35 tasks.

On the same retained paired subsets, we compare Agent execution time and total Token consumption. For each model, we report the mean cost under the Standard and High-Cost conditions together with the corresponding HC/ST cost ratio.

Table 7: After retaining only the paired runs in which tasks were successfully completed under both Standard (ST) and High-Cost (HC) Skill conditions, the execution time and token consumption of the Agent are compared. Valid Pairs indicates the number of run pairs that meet the pairing criteria, and Covered Tasks refers to the number of tasks covered by these valid pairs. HC/ST represents the ratio of the HC average to the ST average for the corresponding metric.

Agent	Pair Coverage		Execution Time (s)			Token Consumption (K)		
	Valid Pairs	Covered Tasks	ST ↓	HC ↓	HC/ST	ST ↓	HC ↓	HC/ST
GPT-5.6-Luna	54/105	25/35	124.21	366.19	2.95	246.07	807.44	3.28
DeepSeek-V4-Flash	64/105	25/35	147.59	395.99	2.68	584.93	851.33	1.46

As shown in Table 7, High-Cost Skills remain associated with higher Agent execution costs even when the comparison is restricted to paired runs in which both Skill conditions successfully complete the task. For GPT-5.6-Luna, the HC/ST ratios are 2.95 for execution time and 3.28 for Token consumption; for DeepSeek-V4-Flash, the corresponding ratios are 2.68 and 1.46. In particular, execution time shows a clear and consistent cost difference across both Agents, while Token consumption follows the same direction but with different magnitudes across models.

Taken together, these two supplementary analyses examine the effects of differences in initial Skill context and of failed runs and timeouts on the cost comparison. Under these controls, the execution-time difference between the Standard and High-Cost Skill conditions remains pronounced, while Token consumption also shows an overall higher-cost pattern under the High-Cost condition. These results further support the intended benchmark construction: both Skill conditions provide effective

guidance for task execution, while corresponding to downstream Agent execution processes with different cost characteristics.

F SUPPLEMENTARY AGENT EXECUTION RESULTS

The preceding evaluation shows that the Cost-Aware Reference Method tends to favor lower-cost Skills at the selection stage. We further examine whether this selection preference can translate into actual downstream efficiency gains. Specifically, we consider two questions: (1) whether cost-aware selection can reduce downstream Agent execution costs while largely preserving task-completion performance, compared with directly using the initial Top-10 retrieval results; and (2) whether the resulting Agent-side savings can still yield end-to-end efficiency gains after accounting for the additional overhead introduced by Cost-Aware Selection. We then evaluate these questions from the perspectives of Agent execution cost and end-to-end cost, respectively.

F.1 EXPERIMENTAL SETUP.

For BM25, Qwen Dense, and SkillRouter, we use the corresponding Top-10 retrieval results as the initial candidate set for the Cost-Aware Reference Method. GPT-5.6-Luna serves as both the backbone of the method and the downstream Agent executor. All downstream execution experiments use the same tasks, tests, Agent configuration, and execution environment.

We compare three Skill-supply variants. TOP-10 directly provides the ten Skills returned by the initial Retriever and serves as the retrieval baseline without further selection. To reduce the effect of randomness in LLM-based Cost-Aware Selection on downstream evaluation, we independently perform Cost-Aware Selection three times for each Top-10 candidate set and take the union of the three selected Skill sets, forming a stabilized evaluation condition denoted as COST-AWARE UNION. We further construct a TOP-M baseline to control for the potential effect of Skill-set cardinality. For each task, M is set to the number of Skills in the corresponding COST-AWARE UNION, and the first M Skills in the original Top-10 ranking are retained. Thus, TOP-M and COST-AWARE UNION have identical Skill-set cardinality for each task while differing in the specific Skills supplied to the Agent.

Table 8 reports the average number of supplied Skills and the resulting initial Skill-context token counts for the three variants. Since TOP-10 and COST-AWARE UNION differ substantially in Skill-set size, their direct comparison may reflect both cost-aware selection and generic Skill-set reduction. TOP-M provides a cardinality-matched control for separating these effects. Moreover, even under matched cardinality, differences in Skill length can lead to different initial context sizes. We therefore report both Full Agent Tokens and Context-Adjusted Agent Tokens in the subsequent evaluation, measuring actual Token consumption and execution-related Token cost after accounting for initial-context differences, respectively.

For each Retriever and execution variant, we evaluate all 35 tasks with three independent Agent executions per task, yielding 105 runs for each condition. All variants use the same Agent executor, task verifier, and execution configuration. The corresponding runtime environment and measurement protocol are described in Appendix D.

Table 8: Average number of supplied Skills and initial Skill-context token count across retrieval and selection variants. Token counts are computed with OpenAI's tiktoken tokenizer using the o200k_base encoding, over the full text of initially supplied SKILL.md files.

Metric	Top-10			Cost-Aware Union			Top-M		
	BM25	Qwen	SkillR.	BM25	Qwen	SkillR.	BM25	Qwen	SkillR.
Avg. # Skills	10.00	10.00	10.00	2.66	2.60	2.83	2.66	2.60	2.83
Initial Tok. (K)	14.53	10.80	13.79	1.45	1.56	1.74	2.13	1.74	1.53

F.2 EVALUATION METRICS.

This subsection defines the additional end-to-end metrics used in the subsequent evaluation. For TOP-10 and TOP-M, the supplied Skill sets are determined directly from the initial Retriever

rankings and therefore introduce no additional LLM-based selection overhead. For COST-AWARE UNION, which is constructed from three independent Cost-Aware Selection runs, we account for the corresponding selection overhead in both Token consumption and execution time.

For Token cost, all three Cost-Aware Selection runs incur independent model calls and Token consumption. We therefore define the selection overhead of COST-AWARE UNION as the sum of the Token consumption of the three runs. For execution time, the three selection runs are independent and can be executed in parallel; we therefore use the maximum execution time among the three runs as the selection latency of COST-AWARE UNION.

Estimated End-to-End Tokens. We estimate end-to-end Token cost by combining Context-Adjusted Agent Tokens with the additional Token overhead introduced by Skill selection:

$$\widehat{\mathrm{Tok}}_{\mathrm{E2E}}(v) = \begin{cases} \mathrm{Avg}_r\left[C_{\mathrm{ctx}}^{(r,v)}\right], & v \in \{\text{TOP-10}, \text{TOP-}M\}, \\ \mathrm{Avg}_r\left[C_{\mathrm{ctx}}^{(r,v)} + \sum_{j=1}^{3} C_{\mathrm{sel},j}^{(r)}\right], & v = \text{COST-AWARE UNION}. \end{cases}$$

Here, $C_{\mathrm{sel},j}^{(r)}$ denotes the Token consumption of the j-th Cost-Aware Selection run. The *Est. E2E Tok. (K)* column reports this metric in thousands of Tokens.

Estimated End-to-End Time. Similarly, we estimate end-to-end execution time as

$$\widehat{T}_{\mathrm{E2E}}(v) = \begin{cases} \mathrm{Avg}_r\left[T_{\mathrm{agent}}^{(r,v)}\right], & v \in \{\text{TOP-10}, \text{TOP-}M\}, \\ \mathrm{Avg}_r\left[T_{\mathrm{agent}}^{(r,v)} + \max_{j\in\{1,2,3\}} T_{\mathrm{sel},j}^{(r)}\right], & v = \text{COST-AWARE UNION}. \end{cases}$$

Here, $T_{\mathrm{sel},j}^{(r)}$ denotes the execution time of the j-th Cost-Aware Selection run. The *Est. E2E Time (s)* column reports this metric.

F.3 EVALUATE RESULTS.

Table 9: Estimated end-to-end performance and agent-execution costs of Top-10 retrieval, Top-M retrieval, and cost-aware selection.

Retr.	Variant	Est. E2E Performance			Agent Execution Cost			
		Pass (%) ↑	$\widehat{\mathrm{Tok}}_{\mathrm{E2E}}$ (K) ↓	$\widehat{T}_{\mathrm{E2E}}$ (s) ↓	Ctx.-adj. Tok. (K) ↓	Full Tok. (K) ↓	Agent Time (s) ↓	Calls ↓
BM25	Top-10	71.43	286.89	248.45	286.89	833.86	248.45	22.29
	Top-M	60.00	301.46	329.05	301.46	540.18	329.05	23.18
	Cost-Aware Union	65.71	184.11	222.04	161.68	323.99	172.73	16.90
Qwen	Top-10	77.14	340.36	263.36	340.36	814.16	263.36	24.90
	Top-M	70.48	279.92	311.85	279.92	528.56	311.85	24.81
	Cost-Aware Union	74.29	250.06	251.41	225.07	403.57	170.12	18.28
SkillRouter	Top-10	68.57	306.47	328.41	306.47	842.04	328.41	23.50
	Top-M	68.57	400.70	373.46	400.70	677.69	373.46	28.40
	Cost-Aware Union	67.62	253.14	228.91	225.76	407.71	180.96	18.22

As shown in Table 9, Cost-Aware Union maintains a Pass Rate broadly comparable to Top-10 and Top-M across all three retrievers, indicating that cost-aware selection does not achieve efficiency gains at the expense of substantial task-performance degradation. At the Agent execution stage, Cost-Aware Union consistently reduces execution cost: compared with Top-10 and Top-M, Agent Time is reduced by up to 51.55% and context-adjusted Token consumption by up to 46.37%, while Full Token consumption and the number of model calls (Calls) are also lower overall. This suggests that, while preserving comparable task-level effectiveness, Cost-Aware Union selects Skill combinations that are more efficient for downstream Agent execution.

We further evaluate the methods from an end-to-end perspective by incorporating the additional overhead introduced by Cost-Aware Selection. After accounting for this selection overhead, Cost-Aware Union still achieves lower estimated end-to-end time $\widehat{T}_{\mathrm{E2E}}$ and Token consumption $\widehat{\mathrm{Tok}}_{\mathrm{E2E}}$ across all three Retrievers. This indicates that the savings achieved during Agent execution are not offset by the additional cost of selection, but instead translate into end-to-end efficiency gains. Overall, Cost-Aware Union reduces downstream Agent execution cost while maintaining a comparable Pass Rate, and these benefits remain after the additional overhead of Cost-Aware Selection is taken into account.