SKILLGATE: GATE-AWARE MONTE CARLO TREE SEARCH FOR SKILL RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

Skill Retrieval (SR) aims to identify the most relevant skills from external skill libraries, and becomes increasingly challenging as libraries grow in scale and diversity. Existing methods either rank skills independently or rely on predefined graph propagation and hierarchical routing, making them vulnerable to semantic distractors, local trapping, and early routing errors. We formulate SR as an adaptive information-foraging process that coordinates region-level navigation with skill-level selection according to the utility and uncertainty observed during search. Based on this formulation, we propose SkillGATE, a graph-guided hierarchical retrieval framework with Gate-Aware Monte Carlo Tree Search (MCTS). SkillGATE constructs a graph-preserving hierarchical index and performs adaptive retrieval through selection, expansion, simulation, and backpropagation. G-PUCT guides action selection, expansion explores new regions, simulation evaluates candidate skills, and backpropagation updates search statistics. Experiments on six SR benchmarks show that SkillGATE consistently improves diverse retrieval and reranking backbones, achieving a 16.3% improvement in overall R@1 over the strongest retriever-based baseline. Our code is available at here.

1 INTRODUCTION

Skill Retrieval aims to identify the most relevant skill from an external skill library for a given query. As skill libraries grow to tens of thousands of entries, directly including all skills in the agent prompt becomes impractical and introduces irrelevant information. Effective SR is therefore essential for accurately retrieving the skill that best matches the query.

Existing approaches broadly fall into two paradigms: Vanilla SR and Structured SR. Vanilla SR treats skills as independent retrieval units and ranks them by query relevance using lexical matching or learned representations. Representative approaches include lexical retrievers (Sparck Jones, 1972; Robertson & Zaragoza, 2009), general-purpose dense retrievers (Karpukhin et al., 2020; Xiao et al., 2024), and embedding models adapted to skill retrieval through task-specific training (Zheng et al., 2026; Cho et al., 2026; Wang et al., 2026b). Structured SR organizes the library into an explicit index structure that guides retrieval through relations or groups of skills. Two common forms are *graph indexing* (Liu et al., 2026; Zeng et al., 2026), which represents inter-skill relations as edges and propagates relevance over the resulting graph, and *hierarchical indexing* (Zeng et al., 2026; Li et al., 2026), which organizes skills into groups and retrieves them through progressively finer routing decisions. Figure 1 provides an intuitive comparison of these retrieval paradigms and their characteristic search behaviors.

However, these approaches exhibit distinct limitations in navigating the skill space. Vanilla SR relies on independent relevance scores, making it susceptible to semantically similar distractors in a large flat search space. Graph-based SR exploits inter-skill relations, but propagation can become confined to local neighborhoods around the initial seeds. Hierarchy-based SR progressively narrows the search space, yet an early routing error may exclude relevant skills with limited opportunities for recovery. These limitations share a common cause: each paradigm operates within a largely fixed search regime and offers limited flexibility to revise its search scope based on evidence gathered during retrieval. Effective SR therefore requires an adaptive search policy that coordinates exploitation of promising neighborhoods, exploration of alternative regions, and pruning of unpro-

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.

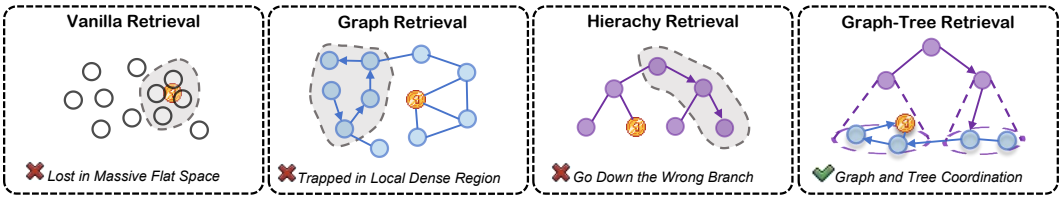

Figure 1: Comparison between different skill retrieval paradigms.

ductive branches. This motivates viewing SR as adaptive information foraging, with the utility and uncertainty of observed returns guiding dynamic adjustments to search scope and strategy.

Motivated by Optimal Foraging Theory (Charnov, 1976; Pyke et al., 1977), we formulate skill retrieval as an adaptive search process over a structured skill space. In this view, different regions of the skill space are treated as *patches*, with individual skills as resources within them. Retrieval uses the observed utility and uncertainty of the current region to guide whether the search should continue locally or move to alternative regions, while prioritizing promising skills within the explored region. This perspective directly addresses the rigidity of existing SR paradigms: rather than committing to a vanilla ranking, local propagation process, or hierarchical routing path, retrieval can dynamically adjust its search scope according to the utility and uncertainty of regions. This enables coordination between region-level navigation and skill-level selection as retrieval evidence accumulates.

Based on this formulation, we propose SKILLGATE, a graph-guided hierarchical retrieval framework with gate-aware Monte Carlo Tree Search. During offline indexing, SKILLGATE constructs a weighted skill graph by integrating semantic, lexical, and structured-label relations, and recursively partitions it into a hierarchical index while preserving skill-level graph connections. This structure supports both progressive region-level navigation and cross-region transitions. During online retrieval, we develop a gate-aware MCTS algorithm that coordinates region-level navigation with skill-level selection through node-aware descent, skill-aware descent, horizontal search, and pruning. Its G-PUCT policy jointly considers historical return, normalized return entropy, query-relevance priors, and visit statistics to adapt the search strategy as retrieval evidence accumulates. Expansion explores promising unvisited regions, simulation evaluates their contained skills and collects relevant candidates, and backpropagation separately updates search visitation and hierarchical reward statistics. Overall, these components enable SKILLGATE to adaptively navigate the structured skill space while maintaining fine-grained relevance to the query.

Our primary contributions are summarized as follows. 1) We formulate skill retrieval as an adaptive information-foraging process that coordinates region-level navigation with skill-level selection. 2) We propose SKILLGATE, a graph-guided hierarchical retrieval framework with gate-aware MCTS for adaptive search and cross-region exploration. 3) Extensive experiments on six SR benchmarks show that SKILLGATE consistently improves diverse retrieval and reranking backbones, achieving a 16.3% improvement in overall R@1 over the strongest retriever-based baseline.

2 RELATED WORK

Skill Retrieval. Existing SR methods can be broadly divided into two paradigms: Vanilla SR, which ranks skills independently through query–skill matching, and Structured SR, which exploits explicit organization or relations among skills. Vanilla SR mainly follows retrieve-and-rerank pipelines. SkillFlow (Li et al., 2025) combines dense retrieval, cross-encoder reranking, and LLM-based selection. SkillRouter (Zheng et al., 2026) leverages full skill content with false-negative filtering and listwise reranking. SkillRet (Cho et al., 2026) provides large-scale benchmarks and retrieval-oriented training data, while R3 (Wang et al., 2026b) uses rejection signals to supervise query-conditioned skill compatibility. Such methods largely operate over independently scored skills or retrieved shortlists. Structured SR explicitly organizes the skill library. AgentSkillOS (Li et al., 2026) builds a capability tree for coarse-to-fine retrieval; Graph-of-Skills (Liu et al., 2026) performs dependency-aware retrieval over a skill graph; and GoSkills (Zeng et al., 2026) organizes skills into graph-connected groups for structured retrieval and context assembly. While these methods demonstrate the value of structural organization, they typically rely on predefined routing or propagation mechanisms. In contrast, SkillGATE combines hierarchical organization with graph connectivity and adaptively coordinates region-level navigation with skill-level selection.

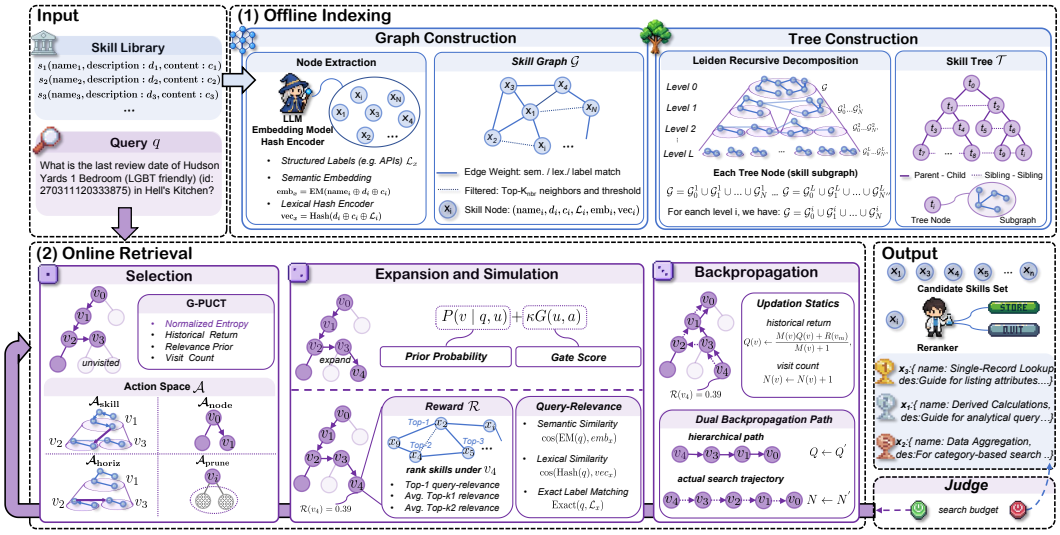

Figure 2: Framework Overview.

Document Retrieval. Document retrieval approaches can be broadly categorized into flat retrieval, which scores individual documents or passages, and structured retrieval, which exploits hierarchies or relational graphs. Early work on flat retrieval primarily relied on lexical methods, such as BM25 (Robertson & Zaragoza, 2009) and TF-IDF (Sparck Jones, 1972), which use vocabulary-based term statistics to estimate relevance through weighted term overlap between queries and documents. More recent efforts have increasingly explored dense retrievers, including Qwen3-Embedding (Zhang et al., 2025) and BGE (Chen et al., 2024). These methods encode queries and documents into a shared dense vector space and rank candidates by embedding similarity, enabling semantic matching beyond exact lexical overlap. Recent efforts along structured retrieval mainly includes: HippoRAGv2 (Gutiérrez et al., 2025), which integrates passages and LLM guidance into Personalized PageRank (PPR) (Jeh & Widom, 2003) retrieval; ComoRAG (Wang et al., 2026a), which constructs a hierarchical knowledge index over raw passages, semantic abstractions, and narrative summaries to support iterative multi-layer retrieval; PonsRAG (Zhao et al., 2026), which connects narrative knowledge across layers through graph-based indexing and coordinates cross-layer retrieval to recover complementary evidence.

3 Overview

In this section, we first formalize the skill retrieval task and then present an overview of our proposed framework, SkillGate.

Problem formulation. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ denote a library of N skills. Each skill s_i is represented by a tuple $(\text{name}_i, d_i, c_i)$, comprising its name, functional description, and detailed execution content, respectively. Given a query q, skill retrieval aims to identify the top-K skills that are most relevant to the query and best suited to its functional requirements. Let F denote an SR framework that takes q and $\mathcal{S}$ as input and returns a ranked list of skills. The resulting top-K skill set is given by

$$\mathcal{S}_K^*(q) = \text{TopK}(F(q, \mathcal{S})) . \tag{1}$$

Our framework. As illustrated in Figure 2, SkillGate consists of *offline indexing* and *online retrieval*. During offline indexing, we first construct a weighted skill graph $\mathcal{G}$, whose nodes represent skills and whose edge weights capture inter-skill relevance. We then recursively partition $\mathcal{G}$ to build a hierarchy in which each non-root node corresponds to a subgraph of $\mathcal{G}$. The resulting index organizes the skill library hierarchically while preserving graph connections to support retrieval both within and across subgraphs. Given a query q, online retrieval employs our gate-aware MCTS to explore the hierarchy and retrieve relevant skills through four stages: selection, expansion, simulation, and backpropagation. Starting from the root, *selection* repeatedly applies the proposed G-PUCT policy until reaching an expandable node, balancing high-reward and low-entropy directions with exploration of less-visited alternatives. *Expansion* adds an unvisited child or sibling node. From the

expanded node, *simulation* follows the default policy until termination or budget exhaustion. Let v_m denote the final node, where the relevance function r evaluates contained skills to produce reward $R(v_m)$, and highly relevant skills are added to the candidate set $\mathcal{S}^*(q)$. During *backpropagation*, we decouple visit-count updates from reward assignment. Visit counts are updated along the search trajectory, while $R(v_m)$ is propagated through the hierarchical path to update node rewards Q. After the prescribed number of iterations, SKILLGATE returns the top-K skills from $\mathcal{S}^*(q)$.

4 METHODOLOGY

4.1 OFFLINE INDEXING

Graph construction. Given a skill $s_i \in \mathcal{S}$, we use an LLM to extract a structured label set $\mathcal{L}_i$ from its tuple representation $(\mathrm{name}_i, d_i, c_i)$. These labels capture the skill's input and output fields, as well as the tools (e.g., APIs) involved in its execution. We then compute a semantic embedding and a lexical hash vector: $\mathrm{emb}_i = \mathrm{EM}(\mathrm{name}_i \oplus d_i \oplus c_i)$ and $\mathrm{vec}_i = \mathrm{Hash}(d_i \oplus c_i \oplus \mathcal{L}_i)$, where $\oplus$ denotes concatenation. Each skill is represented as a graph node $x_i = (\mathrm{name}_i, d_i, c_i, \mathcal{L}_i, \mathrm{emb}_i, \mathrm{vec}_i) \in \mathcal{X}$.

For each pair of skill nodes, we compute an edge weight w_{ij} as a weighted combination of semantic similarity $\cos(\mathrm{emb}_i, \mathrm{emb}_j)$, lexical similarity $\cos(\mathrm{vec}_i, \mathrm{vec}_j)$, and label consistency $\mathrm{Jaccard}(\mathcal{L}_i, \mathcal{L}_j)$. To maintain graph sparsity, we retain, for each node, at most K_{nbr} neighbors with the highest edge weights among those exceeding a threshold τ_e. The resulting weighted skill graph is denoted by $\mathcal{G} = (\mathcal{X}, \mathcal{E})$.

Tree construction. We construct a hierarchical index tree $\mathcal{T}$ whose nodes are subgraphs of the skill graph $\mathcal{G}$, with $\mathcal{G}$ itself as the root. Starting from the root, we recursively apply the Leiden community detection algorithm (Traag et al., 2019) to partition each subgraph into communities, each forming a child node in $\mathcal{T}$. Recursion stops when a node reaches the maximum decomposition depth $l_{\max}$ or contains fewer than $n_{\min}$ skills.

Each tree node t retains the original skill-level edges between its member skills. To obtain node-level representations, we average the semantic embeddings and lexical hash vectors of these skills and normalize the resulting mean vectors, yielding emb_t and vec_t, respectively. We also merge their label sets into a structured label set $\mathcal{L}_t$. Each node is thus a subgraph augmented with the aggregated features $(\mathrm{emb}_t, \mathrm{vec}_t, \mathcal{L}_t)$.

4.2 ONLINE RETRIEVAL

We now describe the online retrieval procedure of SKILLGATE, which consists of selection, expansion, simulation, and backpropagation.

4.2.1 SELECTION

Starting from the root, the selection stage traverses the current MCTS tree by repeatedly choosing the most promising action until an expandable node is reached or a pruning action is selected. At each node u, the search considers four types of actions. *Node-aware descent* ($\mathcal{A}_{\mathrm{node}}$) moves to a child node to further narrow the retrieval scope. *Skill-aware descent* ($\mathcal{A}_{\mathrm{skill}}$) performs PPR within the current subgraph u to identify the highest-scoring skill, and then moves to the child node containing that skill. *Horizontal search* ($\mathcal{A}_{\mathrm{horiz}}$) moves to a sibling node in the index hierarchy through cross-subgraph skill edges, allowing the search to leave a potentially suboptimal branch. *Pruning* ($\mathcal{A}_{\mathrm{prune}}$) excludes the current node and its descendants from further exploration during the current iteration and immediately initiates simulation for reward evaluation. The global action space is therefore defined as $\mathcal{A} = \mathcal{A}_{\mathrm{node}} \cup \mathcal{A}_{\mathrm{skill}} \cup \mathcal{A}_{\mathrm{horiz}} \cup \mathcal{A}_{\mathrm{prune}}$. More details are provided in Appendix B.

To guide action selection, we introduce G-PUCT, which incorporates four signals: historical return, normalized entropy, a query-relevance prior, and visit counts. The historical return Q (Section 4.2.3)reflects the past retrieval performance of a search direction and is normalized to $[0, 1]$. The normalized entropy H characterizes the concentration of a visit-weighted distribution over visited child nodes. The prior probability P (Section 4.2.2) measures the relevance of a candidate node to the query, while the visit counts encourage exploration of less-visited candidates. For an action a

that leads from the current node u to a candidate node v, the G-PUCT score is defined as

$$\text{G-PUCT}(a, v \mid u) = \big(1 - \kappa(u)\big)Q(v) + \kappa(u)G(u, a) + \lambda P(v)\frac{\sqrt{\max(1, N(u))}}{1 + N(v)}, \tag{2}$$

where $N(\cdot)$ denotes the visit count, $G(u, a)$ is an action-specific gate score, and λ controls the strength of prior-guided exploration. The mixing coefficient $\kappa(u)$ is defined as

$$\kappa(u) = 1 - \exp\left(-\frac{N(u)}{\tau_\kappa}\right), \tag{3}$$

where $\tau_\kappa > 0$ controls how quickly the weighting shifts from the candidate node's historical return $Q(v)$ toward the gate score $G(u, a)$. A larger τ_κ slows this transition, preserving the direct contribution of $Q(v)$ over more visits. For pruning actions, we assign sufficiently small values to the return and prior components of the score.

The gate score combines the normalized historical return $Q(u)$ and entropy $H(u)$ as follows:

$$G(u, a) = \begin{cases} Q(u)H(u), & a \in \mathcal{A}_{\text{node}}, \\ Q(u)\big(1 - H(u)\big), & a \in \mathcal{A}_{\text{skill}}, \\ \big(1 - Q(u)\big)H(u), & a \in \mathcal{A}_{\text{horiz}}, \\ \big(1 - Q(u)\big)\big(1 - H(u)\big), & a \in \mathcal{A}_{\text{prune}}. \end{cases} \tag{4}$$

To compute $H(u)$, let $\mathcal{C}(u)$ denote the set of visited children of u. When $\mathcal{C}(u)$ is nonempty, we define a probability distribution over these children by weighting their exponentiated mean returns by their visit counts:

$$p(v_i \mid u) = \frac{N(v_i)\exp(Q(v_i)/\tau_H)}{\sum_{v_j \in \mathcal{C}(u)} N(v_j)\exp(Q(v_j)/\tau_H)}, \tag{5}$$

where $\tau_H > 0$ controls the influence of return differences on the distribution. For nodes with at least two visited children, we normalize the Shannon entropy (Shannon, 1948) by its maximum value:

$$H(u) = -\frac{1}{\log|\mathcal{C}(u)|}\sum_{v_i \in \mathcal{C}(u)} p(v_i \mid u)\log p(v_i \mid u). \tag{6}$$

A higher $H(u)$ indicates a more evenly distributed probability mass, whereas a lower value indicates greater concentration on a smaller subset of children. For a single visited child, we set $H(u) = 0$; when no child has been visited, we initialize $H(u) = 1$ as a convention for the absence of observations. These definitions ensure that $H(u) \in [0, 1]$ and, together with $Q(u) \in [0, 1]$, keep all four gate scores nonnegative. For each feasible action, we compute the corresponding gate and G-PUCT scores and select the action with the highest G-PUCT score.

4.2.2 Expansion and Simulation

The expansion stage begins at an expandable node u and adds an unvisited successor v to the MCTS tree. The choice of v is guided by its query-relevance prior $P(v \mid q, u)$ (also denoted as $P(v)$) and the gate score $\kappa(u)G(u, a)$ of the action a leading to it. To compute the prior $P(v \mid q, u)$, we first define the relevance $r(q, v)$ between the query q and a candidate node v as a weighted combination of semantic similarity $\cos(\text{EM}(q), emb_v)$, lexical similarity $\cos(\text{Hash}(q), vec_v)$, and exact label matching $\text{Exact}(q, \mathcal{L}_v)$. Let $\mathcal{D}(u)$ denote the set of candidate successor nodes reachable from u through feasible non-pruning actions, including eligible child and sibling nodes. We obtain the prior probability by applying a softmax over their relevance scores:

$$P(v \mid q, u) = \frac{\exp(r(q, v)/\tau_P)}{\sum_{v' \in \mathcal{D}(u)} \exp(r(q, v')/\tau_P)}, \tag{7}$$

where $\tau_P > 0$ controls the sharpness of the prior distribution. This distribution is defined over all candidate successors, while expansion is restricted to unvisited ones.

After expansion, the simulation samples feasible non-pruning transitions according to the prior $P(v \mid q, u)$ until no feasible transition remains or the search budget is reached. The utility of this rollout is

evaluated based on the relevance between the query q and the skills contained in the subgraph node v_m. Specifically, we rank these skills using $r(q, x)$ and define the simulation reward $\mathcal{R}(v_m)$ as a weighted sum of three terms: the top-1 relevance score, the average relevance score over the top-k_1 skills, and the average relevance score over the top-k_2 skills. The top-k_2 skills are further added to a global candidate pool.

4.2.3 BACKPROPAGATION

The backpropagation stage updates the statistics used in subsequent MCTS iterations while decoupling search visitation from reward assignment. Let $\mathcal{V}_{\text{visit}}$ denote the actual search trajectory of the current iteration, and let $\mathcal{V}_{\text{hier}}$ denote the hierarchical path from the final node v_m to the root. The visit count is updated along the actual search trajectory: $N(v) \leftarrow N(v) + 1$, where $v \in \mathcal{V}_{\text{visit}}$. In contrast, the simulation reward $R(v_m)$ is propagated along $\mathcal{V}_{\text{hier}}$ to update the historical mean return Q. Since reward assignment is independent of node visitation, we maintain a separate reward count $M(v)$. For each $v \in \mathcal{V}_{\text{hier}}$, the update is

$$Q(v) \leftarrow \frac{M(v)Q(v) + R(v_m)}{M(v) + 1}, \quad M(v) \leftarrow M(v) + 1. \tag{8}$$

Visit and reward counts are initialized as $N(v) = M(v) = 0$. For nodes without observed rewards, we use the neutral calibrated value $Q(v) = 0.5$. This dual-path update mechanism allows N to record the actual exploration process, while Q attributes the retrieved reward to the hierarchical branch producing the relevant skills and its ancestors.

5 EXPERIMENT

In this section, we conduct extensive experiments to evaluate the effectiveness of our method, followed by several analyses to further investigate its key components and behaviors.

5.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate all methods on six benchmarks: TheoremQA (Chen et al., 2023), LogicBench (Parmar et al., 2024), ToolQA (Zhuang et al., 2023), CHAMP (Mao et al., 2024), MedCalc-Bench (Khandekar et al., 2024), and BigCodeBench (Zhuo et al., 2025). All queries are evaluated against the shared 26,262-skill library from SRA-Bench (Su et al., 2026). Following SRA-Bench, we report Recall@1 (R@1) and Recall@10 (R@10) for each benchmark, along with their averages across all benchmarks.

Baselines. We compare against two categories of baselines: retriever-based and reranker-based methods. Retriever-based baselines include general retrievers (BGE-Emb (Chen et al., 2024), Ada-002, and BM25 (Robertson & Zaragoza, 2009)), structured retrievers (GoS (Liu et al., 2026), GoSkills (Zeng et al., 2026), SkillFlow (Li et al., 2025), and AgentSkillOS (Li et al., 2026)), and fine-tuned retrievers (SkillRouter-Emb (Zheng et al., 2026), SkillRet-Emb (Cho et al., 2026), and R3-Emb (Wang et al., 2026b)). Reranker-based baselines augment retrievers with reranking models, including general rerankers (GPT-4o-mini, Qwen3-4B (Yang et al., 2025), and BGE-Rerank (Chen et al., 2024)), structured retrieval methods with reranking, and fine-tuned rerankers from SkillRouter (Zheng et al., 2026) and R3 (Wang et al., 2026b). For structured methods without an original reranker, we use GPT-4o-mini; otherwise, we follow their original settings. Implementation details are provided in Appendix C.

5.2 MAIN RESULTS

To evaluate the effectiveness of our framework, we compare it with two categories of baselines. From results shown in Table 1, we make the following observations: 1) Our framework consistently outperforms existing methods. Under the retriever setting, our method achieves 58.6% R@1 and 88.1% R@10, improving over the strongest baselines by 16.3 and 3.8 percentage points, respectively. Under the reranker setting, it achieves 59.7% R@1 and 86.5% R@10, with gains of 10.8 and 1.6 points. Except for ToolQA and MedCalc., our method improves or matches the best baseline across all benchmarks and metrics. 2) The gains in R@1 are generally larger than those in R@10.

Table 1: Main results (%) on six benchmarks for SR. **Blue** denotes retriever-based methods and **purple** denotes reranker-based methods. **Bold** marks the best in ours, underline marks the best baseline, and ***Improvement*** marks their performance gap.

Method	TheoremQA		LogicBench		ToolQA		CHAMP		MedCalc.		BigCode.		Overall	
	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10
Retriever-based Methods														
BM25	57.2	80.7	12.0	36.1	7.0	55.1	13.2	36.1	29.3	69.2	23.6	61.1	23.0	59.3
BGE-Emb	66.8	86.1	4.1	20.5	32.2	83.4	9.8	34.0	41.4	70.1	20.7	62.1	31.6	65.7
Ada-002	68.0	90.8	9.1	31.6	18.8	54.8	13.7	41.7	82.6	98.4	18.2	52.0	36.9	64.2
GoS	54.8	83.4	7.6	25.5	6.8	20.4	4.9	25.1	67.1	81.8	17.2	49.3	28.0	48.6
GoSkills	36.0	51.1	6.2	10.9	7.0	30.8	5.0	8.6	13.7	25.4	16.3	36.6	14.1	30.0
SkillFlow (emb)	58.8	82.1	1.1	8.0	15.4	51.3	7.7	26.6	32.6	60.5	13.4	41.7	22.2	48.3
SkillRouter-Emb-0.6B	56.1	85.4	5.4	30.7	10.0	34.6	17.3	51.3	71.4	94.9	11.7	39.4	28.9	55.1
SkillRet-Emb-0.6B	61.3	82.9	13.8	56.8	26.9	69.1	6.9	24.6	96.5	100.0	22.6	68.9	42.3	73.7
R3-Emb-0.6B	32.5	75.0	30.3	84.2	34.5	97.9	1.1	20.0	92.3	99.8	22.6	71.2	41.5	84.3
Ours														
+*Ada-002*	74.7	93.8	13.8	54.0	41.4	68.2	17.8	56.1	93.6	**98.1**	23.0	61.1	47.9	73.8
+*SkillRet-Emb-0.6B*	**78.6**	96.4	20.5	74.2	62.5	90.4	**22.7**	61.6	93.6	95.3	26.1	64.6	55.8	83.3
+*R3-Emb-0.6B*	77.9	**97.2**	**33.2**	**87.1**	**65.9**	**92.9**	22.2	**62.6**	**94.1**	97.5	**26.5**	**72.8**	**58.6**	**88.1**
Improvement	*+10.6*	*+6.4*	*+2.9*	*+2.9*	*+31.4*	*-5.0*	*+5.4*	*+11.3*	*-2.4*	*-1.9*	*+2.9*	*+1.6*	*+16.3*	*+3.8*
Reranker-based Methods														
Ada-002+GPT-4o-mini	81.3	93.6	20.4	35.9	33.7	60.0	24.9	52.7	97.2	99.1	19.0	57.3	47.9	68.3
Ada-002+Qwen3-4b	77.1	92.8	17.8	35.8	29.8	59.8	19.3	51.6	87.5	98.5	19.9	56.5	43.9	67.8
BGE-Emb+BGE-Reranker	66.4	86.9	10.8	18.2	39.2	86.9	7.0	28.0	22.3	53.7	7.2	36.7	27.4	57.4
GoS+GPT-4o-mini	75.4	83.4	16.2	27.9	14.7	23.2	16.4	27.4	81.2	82.4	23.8	50.8	38.8	50.2
GoSkills+GPT-4o-mini	50.5	54.2	7.4	10.9	25.1	30.7	7.0	8.6	25.4	25.4	23.3	36.6	25.1	30.4
SkillFlow	69.3	80.7	2.5	3.8	47.9	67.3	24.9	44.6	24.4	42.4	5.4	23.1	29.8	44.9
AgentSkillOS	15.8	18.5	1.5	2.8	15.0	22.2	6.7	17.3	84.0	88.6	2.6	9.9	24.3	29.7
SkillRouter-Rerank-0.6B	83.1	91.8	24.5	40.1	36.9	49.0	28.7	61.5	95.1	96.8	13.4	35.6	48.1	61.1
R3-Rerank-0.6B	78.7	85.1	39.1	90.7	34.5	97.9	20.3	37.5	91.2	99.7	18.7	59.5	48.9	84.9
Ours														
+*GPT-4o-mini*	82.9	95.2	23.0	60.0	48.5	68.0	27.5	64.0	**97.7**	**99.1**	**24.9**	**66.3**	53.9	76.5
+*SkillRouter-Rerank-0.6B*	86.1	96.1	29.6	78.2	62.8	90.1	30.6	69.4	93.5	95.3	15.3	51.1	56.2	81.2
+*R3-Rerank-0.6B*	**87.7**	**97.3**	**39.9**	**93.3**	**72.5**	**92.4**	**37.0**	**70.5**	85.0	97.6	18.7	59.9	**59.7**	**86.5**
Improvement	*+4.6*	*+3.7*	*+0.8*	*+2.6*	*+24.6*	*-5.5*	*+8.3*	*+9.0*	*+0.5*	*-0.6*	*+1.1*	*+6.8*	*+10.8*	*+1.6*

The overall R@1 improves by 16.3 and 10.8 points under the retriever and reranker settings, respectively, while the corresponding R@10 gains are 3.8 and 1.6 points. This suggests that our framework is particularly effective at ranking the most relevant skill at the top. In particular, on ToolQA, the retriever and reranker settings improve R@1 by 31.4 and 24.6 points, respectively, while reducing R@10 by 5.0 and 5.5 points. This is because the candidate skills in ToolQA are highly similar, and our more focused branch selection improves top-1 precision but may miss the correct branch, thereby reducing Top-10 coverage. 3) Our framework generalizes well across different backbones. With Ada-002, SkillRet-Embedding, and R3-Embedding, our method achieves overall R@1/R@10 scores of 47.9/73.8, 55.8/83.3, and 58.6/88.1, respectively, all outperforming their corresponding embedding-based baselines. Similar gains are observed with GPT-4o-mini, SkillRouter-Reranker, and R3-Reranker, showing that the framework does not depend on a specific backbone. 4) ToolQA shows the largest R@1 gains, while CHAMP shows the largest R@10 gains. On ToolQA, R@1 improves by 31.4 and 24.6 points under the retriever and reranker settings, respectively, which is consistent with the need to distinguish among functionally similar tool skills. In contrast, CHAMP achieves the largest R@10 gains, with improvements of 11.3 and 9.0 points, suggesting better coverage of multiple relevant mathematical skills.

5.3 Ablation Study

To evaluate the contribution of each component in our framework, we conduct ablation studies using Ada-002 without rerankers. For the indexing ablations, the graph-only variant performs PPR directly over the skill graph, while the tree-only variant performs vanilla MCTS over the hierarchical tree. For the retrieval ablations, we remove each proposed component individually while keeping the remaining settings unchanged. Specifically, w/o Reward replaces the proposed reward with the

Table 2: Ablation study on the indexing and retrieval components of our framework.

Method	TheoremQA		LogicBench		ToolQA		CHAMP		MedCalc.		BigCode.		Overall	
	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10
Ours	74.7	93.8	13.8	54.0	41.4	68.2	17.8	56.1	93.6	98.1	23.0	61.1	47.9	73.8
Indexing														
w/ Graph $\mathcal{G}$	58.9	74.6	15.3	51.3	8.8	14.1	15.4	50.4	70.4	80.5	10.8	25.8	29.9	45.3
w/ Tree $\mathcal{T}$	24.2	29.2	4.5	11.7	4.7	5.6	7.8	21.6	36.1	36.9	4.3	7.9	13.8	17.3
Retrieval														
w/o $\mathcal{A}_{\text{skill}}$	47.7	56.9	11.3	29.6	24.9	39.8	17.3	43.2	65.9	72.2	15.8	30.2	32.2	45.4
w/o $\mathcal{A}_{\text{horiz}}$	66.8	84.4	14.3	52.1	33.8	59.0	15.5	44.8	82.5	87.5	17.3	50.7	41.3	65.0
w/o $\mathcal{A}_{\text{prune}}$	74.1	89.2	14.3	50.5	42.5	70.2	17.5	51.6	90.0	93.1	20.5	58.4	46.9	71.5
w/o Gate G	64.8	85.6	12.1	36.3	34.5	58.2	14.4	48.6	87.8	95.2	20.8	60.2	42.6	66.4
w/o Reward	74.2	90.0	14.2	53.3	41.5	67.3	17.2	51.2	89.4	93.4	20.3	56.7	46.5	70.9
w/o $\mathcal{V}_{\text{hier}}$	74.0	89.7	14.7	51.6	41.6	62.1	17.3	51.6	87.0	90.9	20.3	57.4	46.0	68.9
w/o $\mathcal{V}_{\text{visit}}$	74.0	89.6	14.2	51.2	41.2	65.4	16.5	49.3	87.0	90.9	19.8	51.2	45.7	68.3

average relevance over all skills in the final node; w/o $\mathcal{V}_{\text{hier}}$ and w/o $\mathcal{V}_{\text{visit}}$ collapse the decoupled backpropagation into a single update path; the remaining variants remove $\mathcal{A}_{\text{skill}}$, $\mathcal{A}_{\text{horiz}}$, or $\mathcal{A}_{\text{prune}}$, or replace G-PUCT with vanilla PUCT. The results are reported in Table 2. Overall, all components contribute positively to performance. The full model achieves 47.9% R@1 and 73.8% R@10, while using only the graph or tree reduces performance to 29.9/45.3% and 13.8/17.3%, respectively, showing their complementary roles in narrowing the search space and capturing relevant skills beyond a single hierarchical path. Among the retrieval components, removing $\mathcal{A}_{\text{skill}}$ causes the largest drop of 15.7/28.4 points, followed by $\mathcal{A}_{\text{horiz}}$ with 6.6/8.8 points, highlighting the importance of skill-aware descent and horizontal transitions. Replacing G-PUCT with vanilla PUCT decreases R@1/R@10 by 5.3/7.4 points, confirming the benefit of the gate-aware selection strategy. The reward design and decoupled backpropagation also provide consistent gains, with removing the reward, $\mathcal{V}$hier, and $\mathcal{V}$visit causing drops of 1.4/2.9, 1.9/4.9, and 2.2/5.5 points, respectively.

5.4 Detail Analysis

Next, we conduct a detailed analysis of the key components and behaviors of our framework.

Effectiveness of Gate-aware MCTS Actions. Our gate-aware MCTS includes four actions: $\mathcal{A}_{\text{node}}$, $\mathcal{A}_{\text{skill}}$, $\mathcal{A}_{\text{horiz}}$, and $\mathcal{A}_{\text{prune}}$, where the latter three extend vanilla MCTS. To evaluate their effectiveness, we analyze these actions in Table 3. Level denotes the tree depth from L0 to L4; Nodes is the number of nodes at each level; Avg. Skills is the average number of skills per node; Avg. Child is the average number of children per non-leaf node; and Avg. Sim measures the average similarity among sibling nodes based on their representations $t = (emb_t, vec_t, \mathcal{L}_t)$. $\mathcal{A}_{\text{node}}$ Fail denotes the fraction of queries that are first routed incorrectly at the current level. $\mathcal{A}_{\text{skill}}$ Rescue and $\mathcal{A}_{\text{horiz}}$ Rescue denote the fractions of these failures recovered by applying $\mathcal{A}_{\text{skill}}$ from the parent node and $\mathcal{A}_{\text{horiz}}$ from the failed node, respectively. $\mathcal{A}_{\text{prune}}$ Frequency denotes the proportion of selected pruning actions, while $\mathcal{A}_{\text{prune}}$ Acc measures the proportion that correctly remove branches without the gold skill. From the results, we make four observations. First, the hierarchy progressively narrows the retrieval space. The average number of skills per node decreases from 625.29 at L1 to 12.01 at L4, while Avg. Child drops from 42.00 to 3.21. Meanwhile, Avg. Sim increases from 0.22 to 0.43, indicating that deeper levels contain fewer but more fine-grained and similar nodes. Second, $\mathcal{A}_{\text{node}}$ failures mainly arise from the large search width at shallow levels and fine-grained ambiguity at deeper levels. The failure rate is highest at L1 (30.56%), decreases at L2 and L3 as the search space narrows, and rises again to 20.22% at L4, where sibling nodes become more similar. Third, $\mathcal{A}_{\text{skill}}$ and $\mathcal{A}_{\text{horiz}}$ effectively recover errors introduced by $\mathcal{A}_{\text{node}}$. Once $\mathcal{A}_{\text{node}}$ routes to an incorrect branch, $\mathcal{A}_{\text{skill}}$ and $\mathcal{A}_{\text{horiz}}$ successfully redirect the search to a gold-containing branch in 41.67%-74.51% and 35.15%-64.63% of the failed cases, respectively, demonstrating their complementary roles in branch refinement and cross-branch correction. Finally, $\mathcal{A}_{\text{prune}}$ is conservative but reliable. It is selected in only 2.35%-6.45% of cases, while $\mathcal{A}_{\text{prune}}$ Acc remains consistently high at 94.97%-96.19%.

Effectiveness of the Gate in G-PUCT. We further examine whether the gate improves action selection and whether the node-specific entropy H provides useful search signals. We record node states at iterations 16, 32, 64, 96, and 128, and compare four strategies from each identical state: PUCT

Table 3: Hierachy statics and level-wise analysis of actions.

Hierarchy					Action Recovery			Prune Safety	
Level	Nodes	Avg. Skills	Avg. Chid	Avg. Sim	$\mathcal{A}_{\text{node}}$ Fail	$\mathcal{A}_{\text{skill}}$ Res.	$\mathcal{A}_{\text{horiz}}$ Res.	$\mathcal{A}_{\text{prune}}$ Freq.	$\mathcal{A}_{\text{prune}}$ Acc.
L0	1	26,262.00	42.00	–	0.00%	0.00%	0.00%	0.00%	–
L1	42	625.29	8.67	0.22	30.56%	64.85%	35.15%	2.53%	95.49%
L2	338	77.52	4.30	0.32	15.19%	47.56%	64.63%	2.35%	95.46%
L3	1,126	21.17	3.21	0.41	10.16%	74.51%	58.82%	4.48%	94.97%
L4	880	12.01	0.00	0.43	20.22%	41.67%	47.22%	6.45%	96.19%

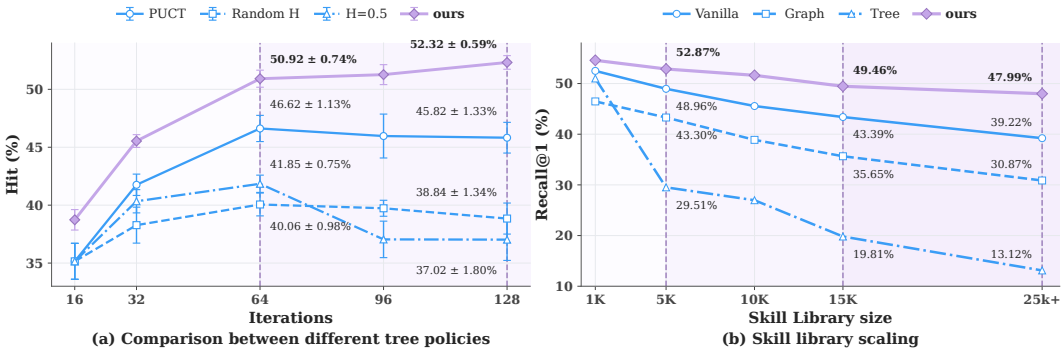

Figure 3: Gate effectiveness (a) and skill-library (b) scalability.

without the gate, G-PUCT with the observed Q and H, Random H with randomized entropy, and H=0.5 with fixed entropy. An action is counted as a hit if its target node contains the gold skill. All methods are evaluated on the same valid node set and repeated with five MCTS seeds, reporting mean and standard deviation. The results are reported in Figure 3(a). From the results, we make two observations. First, G-PUCT consistently outperforms PUCT, with the gap widening as search progresses. At iteration 64, G-PUCT achieves 50.92% Hit versus 46.62% for PUCT; at iteration 128, the gap grows to 6.50% (52.32% vs. 45.82%). This indicates that the gate increasingly benefits from accumulated search statistics. Second, the observed entropy H is critical for effective gating. At iteration 128, Random H and H=0.5 achieve only 38.84% and 37.02%, respectively, far below G-PUCT. Thus, the gains arise not merely from adding a gate term, but from using node-specific uncertainty to adapt action preferences.

Scalability to large skill libraries. To evaluate the scalability of different retrieval methods, we progressively enlarge the skill library from 1K to 5K, 10K, 15K, and the full 26,262 skills, and report Recall@1 at each scale. As the library grows, more irrelevant skills are introduced, making it increasingly difficult to identify the most relevant skill. We compare four methods: Vanilla, which ranks skills by weighted similarity; Graph, which performs PPR over the global skill graph; Tree, which applies MCTS over the hierarchical tree; and Ours, which combines hierarchical routing with graph-based search. As shown in Figure 3(b), all methods degrade as the skill library grows, but Ours consistently achieves the highest Recall@1 and the smallest performance drop. From 1K to the full library, Vanilla and Graph decrease by 10.38 and 15.60 percentage points, respectively, while Tree suffers a much larger drop of 37.88 points. In contrast, Ours remains relatively stable across different library sizes. These results suggest that similarity-based retrieval is increasingly affected by distractors, global graph search becomes more susceptible to noise as the graph expands, and tree-only search tends to accumulate routing errors. By progressively narrowing the search space through the hierarchy and using graph actions to recover from incorrect routes, our full framework scales more robustly to large skill libraries.

6 Conclusion

We introduced SkillGATE, a graph-guided hierarchical framework for skill retrieval with Gate-Aware MCTS. Inspired by Optimal Foraging Theory, SkillGATE formulates retrieval as an adaptive process that coordinates region-level navigation with skill-level selection over a graph-preserving hierarchy. Experiments demonstrate consistent improvements across retrieval and reranking backbones, while analyses validate the effectiveness of the proposed search mechanism.

AI USE STATEMENT

We used generative AI tools to assist with standardizing mathematical notation and the presentation of equations, testing and evaluating code, and improving the clarity and readability of the manuscript. The modeling approach and core methodology were developed by the authors without generative AI assistance. The authors reviewed all AI-assisted material, checked the mathematical expressions, and verified the code evaluation results. We take full responsibility for the accuracy and integrity of the final paper, including all text, claims, and results.

REFERENCES

Eric L Charnov. Optimal foraging, the marginal value theorem. *Theoretical population biology*, 9 (2):129–136, 1976.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the association for computational linguistics: ACL 2024*, pp. 2318–2335, 2024.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. Theoremqa: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901, 2023.

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

Fan Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pp. 271–279, 2003.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W Safranek, Abid A Anwar, Andrew Zhang, et al. Medcalc-bench: Evaluating large language models for medical calculations. *Advances in Neural Information Processing Systems*, 37:84730–84745, 2024.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pp. 282–293. Springer, 2006. doi: 10.1007/11871842_29.

Fangzhou Li, Pagkratios Tagkopoulos, and Ilias Tagkopoulos. Skillflow: Scalable and efficient agent skill retrieval system. *arXiv preprint arXiv:2504.06188*, 2025.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. *arXiv preprint arXiv:2603.02176*, 2026.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026.

László Lovász. Random walks on graphs: A survey. In *Combinatorics, Paul Erdős is Eighty*, volume 2, pp. 1–46. 1993.

Yujun Mao, Yoon Kim, and Yilun Zhou. Champ: A competition-level dataset for fine-grained analyses of llms' mathematical reasoning capabilities. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13256–13274, 2024.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707, 2024.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

Graham H. Pyke, H. Ronald Pulliam, and Eric L. Charnov. Optimal foraging: A selective review of theory and tests. *The Quarterly Review of Biology*, 52(2):137–154, 1977. doi: 10.1086/409852.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. doi: 10.1038/nature16961.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic ai. *arXiv preprint arXiv:2604.24594*, 2026.

Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019. doi: 10.1038/s41598-019-41695-z.

Juyuan Wang, Rongchen Zhao, Wei Wei, Yufeng Wang, Mo Yu, Jie Zhou, Jin Xu, and Liyan Xu. Comorag: A cognitive-inspired memory-organized rag for stateful long narrative reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33557–33565, 2026a.

Zifei Wang, Wei Wen, Qiang Ji, Ruizhi Qiao, and Xing Sun. Skill is not document: A query-conditional benchmark and two-stage retriever for llm agent skill routing. *arXiv preprint arXiv:2606.03565*, 2026b.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 641–649, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Kun Zeng, Yu Huo, Siyu Zhang, Zi Ye, Yuecheng Zhuo, Haoyue Liu, Yuquan Lu, Junhao Wen, and Xiaoying Tang. Group of skills: Group-structured skill retrieval for agent skill libraries. *arXiv preprint arXiv:2605.06978*, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. URL https://arxiv.org/abs/2506.05176.

Rongchen Zhao, Yu Chen, Juyuan Wang, Zhouting Mo, Jianxing Yu, Wenqing Chen, and Jingping Liu. Ponsrag: A pons-inspired rag bridging cognitive islands for coordinated long narrative reasoning. *arXiv preprint arXiv:2608.25486*, 2026.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, et al. Skillrouter: Skill routing for llm agents at scale. *arXiv preprint arXiv:2603.22455*, 2026.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36: 50117–50143, 2023.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

A ALGORITHM

In this section, we present the algorithms for *offline indexing* (Algorithm 1) and *online retrieval* (Algorithm 2).

Algorithm 1 OFFLINE INDEXING

Require: Skill library $\mathcal{S}$; neighborhood size K_{nbr}; edge threshold τ_e; maximum depth $l_{\max}$; minimum node size $n_{\min}$
Ensure: Skill graph $\mathcal{G}$ and hierarchical index $\mathcal{T}$

function BUILDINDEX($\mathcal{S}$)
  $\mathcal{X} \leftarrow$ ENCODE-SKILLS($\mathcal{S}$) ▷ Labels, embeddings, and lexical vectors
  $\mathcal{G} \leftarrow$ **Graph Construction**$(\mathcal{X}, K_{\text{nbr}}, \tau_e)$ ▷ Weighted skill graph
  $\mathcal{T} \leftarrow$ **Tree Construction**$(\mathcal{G}, l_{\max}, n_{\min})$ ▷ Recursive Leiden decomposition
  for $u \in \mathcal{T}$ **do**
    $(\text{emb}_u, \text{vec}_u, \mathcal{L}_u) \leftarrow$ AGGREGATE(Skills(u))
  end for
  return $(\mathcal{G}, \mathcal{T})$
end function

Algorithm 2 ONLINE RETRIEVAL

Require: Query q; skill graph $\mathcal{G}$; hierarchical index $\mathcal{T}$; iteration budget T; rollout budget B; reward cutoffs k_1, k_2; output size K
Ensure: Retrieved skills $\mathcal{S}^*_K(q)$

function RETRIEVE($q, \mathcal{G}, \mathcal{T}, T, B, K$)
  $\mathcal{S}^*(q) \leftarrow \varnothing$; $\mathcal{T}_{\text{search}} \leftarrow \{\text{Root}(\mathcal{T})\}$
  $N(v), M(v) \leftarrow 0, 0$; $Q(v) \leftarrow 0.5, \forall v \in \mathcal{T}$ ▷ Neutral value before reward observation
  for $t = 1, \ldots, T$ **do**
    $(u, a, \mathcal{V}_{\text{visit}}) \leftarrow$ **Selection**$(q, \mathcal{G}, \mathcal{T}, \mathcal{T}_{\text{search}}, N, Q)$ ▷ G-PUCT over expanded transitions
    if $a \neq \mathcal{A}_{\text{prune}}$ **then**
      $v \leftarrow$ **Expansion**$(q, u, a, \mathcal{G}, \mathcal{T})$ ▷ Best unexpanded successor under a
      $v = \arg \max_{\substack{v' \in \mathcal{D}(u),\ v' \notin \mathcal{T}_{\text{search}} \\ a \rightsquigarrow v'}} \left[P(v' \mid q, u) + 0.4\,\kappa(u) G(u, a)\right]$
      Add v and transition (u, v) to $\mathcal{T}_{\text{search}}$
      $u \leftarrow v$; append u to $\mathcal{V}_{\text{visit}}$
      Simulation : $\pi \leftarrow [\,]$
      for $\ell = 1, \ldots, B$ **do**
        if $\mathcal{D}(u) = \varnothing$ **then**
          break
        end if
        $v \sim P(\cdot \mid q, u), v \in \mathcal{D}(u)$ ▷ Query-prior sampling
        $a \leftarrow \text{Action}(u, v)$
        append (u, a, v) to π; $u \leftarrow v$
      end for
      $v_m \leftarrow u$
      $(v_m, \pi) \leftarrow \text{ReadoutFallback}(u, \pi)$ ▷ Ensure a valid final readout
      Append rollout nodes in π to $\mathcal{V}_{\text{visit}}$
    else
      $v_m \leftarrow u$ ▷ Terminate the current trajectory
    end if
    $(R(v_m), \mathcal{C}_t) \leftarrow$ EVALUATE(q, v_m, r, k_1, k_2) ▷ Reward and top-k_2 candidates
    $\mathcal{S}^*(q) \leftarrow \mathcal{S}^*(q) \cup \mathcal{C}_t$
    Backpropagation : $N(v) \leftarrow N(v) + 1, \forall v \in \mathcal{V}_{\text{visit}}$ ▷ Actual search trajectory
    $\mathcal{V}_{\text{hier}} \leftarrow$ ANCESTORPATH$(v_m, \mathcal{T})$
    for $v \in \mathcal{V}_{\text{hier}}$ **do**
      $Q(v) \leftarrow \dfrac{M(v)Q(v) + R(v_m)}{M(v) + 1}$; $M(v) \leftarrow M(v) + 1$
    end for
  end for
  return $\text{TopK}_{x \in \mathcal{S}^*(q)}\, r(q, x)$
end function

B GRAPH-BASED SEARCH ACTIONS DETAILS

In this section, we detail the two graph-based actions, skill-aware descend ($\mathcal{A}_{\text{skill}}$) and horizontal search ($\mathcal{A}_{\text{horizon}}$). Both actions share a query-guided PPR procedure but differ in their propagation scopes and target selection rules.

PPR Initialization. Within the subgraph $\mathcal{G}_u$ associated with the current node u, we select the eight skill nodes with the highest query–skill relevance scores $r(q, x)$ as seeds. Let $\mathcal{S}_{\text{seed}}(u)$ denote the resulting seed set. We normalize their relevance scores to obtain the restart distribution used to initialize PPR:

$$p_0^u(x) = \begin{cases} \dfrac{r(q,x)}{\sum_{x' \in \mathcal{S}_{\text{seed}}(u)} r(q,x')}, & x \in \mathcal{S}_{\text{seed}}(u), \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Skill-aware Descend. For $\mathcal{A}_{\text{skill}}$, we restrict PPR propagation to $\mathcal{G}_u$, normalizing outgoing edge weights over neighbors within this subgraph. Let $\pi_{\text{intra}}(x \mid q, u)$ denote the resulting PPR score, we select the highest-scoring skill node x^*:

$$x^* = \arg\max_{x \in \mathcal{G}_u(u)} \pi_{\text{intra}}(x \mid q, u), \tag{10}$$

and descend to the child of u whose subgraph contains x^*. Thus, the descent direction is determined by skill-level evidence obtained through intra-subgraph propagation.

Horizontal Search. For $\mathcal{A}_{\text{horizon}}$, we use the same seeds but perform PPR over the full skill graph $\mathcal{G}$, allowing relevance signals to reach external skills through cross-subgraph edges. Let $\pi_{\text{inter}}(x \mid q, u)$ denote the resulting PPR score. We group retained external skills by their corresponding sibling subgraphs. For each sibling node $v \in \text{Sib}(u)$, we select the target node v^* by cumulative PPR score:

$$v^* = \arg\max_{v \in \text{Sib}(u)} \sum_{x \in \mathcal{G}_v} \pi_{\text{inter}}(x \mid q, u). \tag{11}$$

To prevent oscillation between sibling nodes, we exclude nodes already visited in the current MCTS iteration from horizontal target selection.

C IMPLEMENTATION DETAILS.

We use GPT-4o-mini as the LLM backbone with a temperature of 0.5 and TF-IDF as the hash encoder. We evaluate our framework under three embedding–reranker configurations: Ada-002 with GPT-4o-mini, SkillRet-Emb-0.6B with SkillRouter-Rerank-0.6B, and R3-Emb-0.6B with R3-Rerank-0.6B. Hyperparameter setting details are provided in Appendix C.1. Following ComoRAG (Wang et al., 2026a), we further conduct all the analysis on a sampled set detailed in Appendix D.

C.1 HYPERPARAMETER SETTINGS

We tune the hyperparameters via grid search on a separate dataset, SkillRet (Cho et al., 2026), and keep the selected configuration fixed for evaluation on SRA-Bench. Table 4 summarizes the hyperparameter settings, organized by the two main stages of SKILLGATE: offline indexing and online retrieval. These settings cover graph and hierarchy construction, gate-aware search, reward estimation, and retrieval budgets. All other settings follow the original configurations. We further analyze the sensitivity of key hyperparameters on the sampled test subsets in the following section.

Table 4: Hyperparameter settings.

Stage	Hyperparameter	Value
Index	Edge weights: sem./lex./label	0.35/0.20/0.45
	Neighbor limit K_{nbr}	50
	Edge threshold τ_e	0.32
	Depth $l_{\max}$ / Number $n_{\min}$	4/20
Retrieval	Relevance weights: sem./lex./label	0.50/0.25/0.25
	Gate Transition scale τ_κ	16
	Entropy temperature τ_H / Prior temperature τ_P	0.20/0.90
	Reward: k_1/k_2	5/10
	Reward weights: top-1/top-k_1/top-k_2	0.55/0.30/0.15
	Maximum iterations/Final output K/Search budget	128/50/16

C.2 Sensitivity Analysis

We further investigate the sensitivity of SkillGate to the query-relevance and reward weights, using Ada-002 as the embedding model without reranking. Specifically, we vary one group of weights at a time while keeping all other settings fixed. Results are reported in Table 5.

Query-relevance weights. The results suggest that standalone signal strength does not directly determine its contribution to the combined relevance score. Although lexical relevance performs best among the single-signal settings (blue rows), the best-performing configuration $(0.50, 0.25, 0.25)$ assigns the largest weight to semantic relevance. Moreover, all single-signal settings yield lower Recall@10 than this combination, supporting the value of integrating complementary evidence rather than relying exclusively on the strongest individual signal. The mixed-weight settings (purple rows) exhibit variations of only 0.36 and 0.60 percentage points in Recall@1 and Recall@10, respectively, suggesting that this combination allows flexibility in weighting rather than requiring a narrowly tuned configuration.

Reward weights. The results highlight the distinction between identifying a high-scoring skill and estimating the utility of an entire search branch. Although the top-1 term receives the largest weight in the default configuration $(0.55, 0.30, 0.15)$, it performs worst when used alone (blue rows). Combining it with the top-k_1 and top-k_2 terms achieves the best performance, suggesting that the strongest individual relevance score is more useful when supported by evidence from multiple highly ranked candidates. Thus, effective reward estimation should emphasize the best candidate without treating its score as sufficient evidence of branch quality. The limited variations across mixed-weight settings (purple rows)—0.41 and 0.57 percentage points in Recall@1 and Recall@10, respectively—further suggest that this benefit is not highly sensitive to the precise weighting within the evaluated range.

Table 5: Sensitivity to query-relevance and reward weights.

Query-Relevance Weights					**Reward Weights**				
Relevance Weights			**Retrieval**		**Reward Weights**			**Retrieval**	
sem.	lex.	label	R@1	R@10	top-1	top-k_1	top-k_2	R@1	R@10
0.50	**0.25**	**0.25**	$\mathbf{47.98 \pm 0.12}$	$\mathbf{74.46 \pm 0.32}$	**0.55**	**0.30**	**0.15**	$\mathbf{47.98 \pm 0.12}$	$\mathbf{74.46 \pm 0.32}$
0.00	0.00	1.00	47.47 ± 0.45	72.49 ± 0.63	0.00	0.00	1.00	47.51 ± 0.40	73.76 ± 0.36
0.00	1.00	0.00	47.54 ± 0.31	73.16 ± 0.56	0.00	1.00	0.00	47.22 ± 0.33	73.70 ± 0.64
1.00	0.00	0.00	46.83 ± 0.14	71.74 ± 0.59	1.00	0.00	0.00	46.80 ± 0.28	72.80 ± 0.26
0.33	0.33	0.33	47.47 ± 0.28	73.45 ± 0.56	0.33	0.33	0.33	47.22 ± 0.41	73.64 ± 0.52
0.25	0.50	0.25	47.15 ± 0.44	73.32 ± 0.41	0.25	0.50	0.25	47.27 ± 0.34	73.87 ± 0.18
0.25	0.25	0.50	47.51 ± 0.42	73.92 ± 0.49	0.25	0.25	0.50	46.86 ± 0.51	73.30 ± 0.36

D Sampled Set Details

We randomly sample 10% of the queries from each of the six benchmarks using a fixed random seed, yielding a subset of 540 queries. Only the query set is subsampled; the complete library of 26,262 skills remains unchanged. To assess whether this subset yields performance comparable to the full benchmark, we report the results of SKILLGATE on the subset using Ada-002 without reranking. As Tabel 6 shown, SKILLGATE achieves 47.98% R@1 and 74.46% R@10 on the subset, compared with 47.88% and 73.83%, respectively, on the full benchmark (Table 1). The respective differences of 0.10 % and 0.63% indicate comparable SR performance between the sampled subset and the full benchmark.

Table 6: Sampled test set statistics.

Dataset	Query	Gold	R@1	R@10
BigCodeBench	114	311	23.92	63.08
CHAMP	22	36	12.88	41.67
LogicBench	76	76	14.47	56.58
MedCalcBench	110	110	93.64	97.27
TheoremQA	75	75	68.00	90.67
ToolQA	143	143	44.76	72.03
Overall	**540**	**751**	**47.98**	**74.46**

E ANALYSIS OF SEARCH STRATEGIES

To evaluate the effectiveness of our search strategy, we compare G-PUCT with six alternatives on the same index. The blue rows correspond to graph propagation methods: Random Walk (RW) (Lovász, 1993), which samples transitions between neighboring skill nodes according to the graph's transition probabilities; Heat Kernel Diffusion (HKD) (Chung, 2007), which aggregates neighborhood information across different propagation lengths through heat-kernel diffusion; and Personalized PageRank (PPR) (Jeh & Widom, 2003), which incorporates restarts to the seed distribution to maintain the influence of seed nodes during propagation. The purple rows correspond to hierachy-based search strategies: Greedy (Pearl, 1984), which selects the child node with the highest prior probability $P(v)$ at each step; UCT (Kocsis & Szepesvári, 2006), which combines historical mean returns with a visit-count-based exploration bonus to balance exploiting promising branches and exploring less-visited ones; and PUCT (Silver et al., 2016), which additionally weights the exploration bonus by the prior probability $P(v)$ to guide node selection.

From Table 7, we yields three observations: 1) Improving graph propagation alone yields limited gains, highlighting the value of hierarchical guidance. RW, HKD, and PPR achieve similar performance: replacing RW with PPR improves R@1/R@10 by only 0.99/1.24 percentage points. In contrast, even without search feedback, hierarchy-based Greedy achieves 36.93/59.80, outperforming PPR at 31.04/47.82. These results suggest that the limitations of graph-only retrieval cannot be fully addressed by changing the propagation rule alone; organizing the search space through hierarchical guidance is also important. 2) Combining hierarchical guidance with graph relations and search feedback improves upon prior-only descent. Both UCT and PUCT outperform Greedy, which selects children solely according to their priors. In particular, PUCT improves R@1/R@10 from 36.93/59.80 to 43.73/67.62. This suggests that hierarchical organization provides a useful starting point, but effective retrieval further benefits from incorporating relational evidence and accumulated search feedback rather than relying exclusively on node priors. 3) How actions are selected matters beyond which search components are available. G-PUCT, UCT, and PUCT all exploit hierarchy, graph relations, and search feedback, yet G-PUCT achieves the highest R@1/R@10 of 47.98/74.46, exceeding the strongest alternative, PUCT, by 4.25/6.84 percentage points. These gains support the value of gate-aware action selection: providing access to complementary search operations is not sufficient without an effective policy for deciding when to use them. The larger improvement in R@10 further indicates that the benefits extend beyond the top-ranked result to broader coverage of relevant skills within the top ten results.

Table 7: Comparison of search strategies on the sampled subset.

Search Strategy	Hierarchy	Graph	Feedback	R@1	R@10
RW	×	✓	×	30.05	46.58
HKD	×	✓	×	30.52	47.06
PPR	×	✓	×	31.04	47.82
Greedy	✓	×	×	36.93	59.80
UCT	✓	✓	✓	42.16	65.51
PUCT	✓	✓	✓	43.73	67.62
G-PUCT	✓	✓	✓	**47.98**	**74.46**

F In-depth Analysis of Decoupled Backpropagation

We compare our decoupled backpropagation with two coupled variants that update both the historical return Q and visit count N along a single path: either the hierarchical parent path $\mathcal{V}_{\text{hier}}$ or the actual visited path $\mathcal{V}_{\text{visit}}$.

Table 8: Update-path comparison.

Level	$\mathcal{V}_{\text{visit}}$ Hit	$\mathcal{V}_{\text{hier}}$ Hit	Ours Hit
L0	100.0	100.0	100.0
L1	90.4	63.0	90.4
L2	64.0	39.2	66.8
L3	40.2	23.7	42.0
L4	28.0	16.3	32.2

From Table 8, we make two observations. First, $\mathcal{V}_{\text{hier}}$ yields the lowest hit rate at every non-root level, decreasing from 63.0% at L1 to 16.3% at L4. Its substantial performance gap relative to the other two variants suggests that visit counts should reflect the actual search trajectory rather than hierarchical ancestry. Second, $\mathcal{V}_{\text{visit}}$ is more competitive and matches our method at L1 (90.4%), but decoupled backpropagation achieves higher hit rates at all subsequent levels. Specifically, our method improves over $\mathcal{V}_{\text{visit}}$ by 2.8, 1.8, and 4.2 percentage points at L2, L3, and L4, respectively, with the largest improvement at the deepest level. This suggests that faithfully tracking visits alone is insufficient: reward attribution also benefits from following the hierarchy of the final node. Together, these results support separating visitation tracking from reward attribution rather than updating both statistics along a single path.

G In-depth Analysis of Efficiency

We compare SkillGATE with the graph-based SR method GoS (Liu et al., 2026) and the hierarchy-based SR method AgentSkillOS (Li et al., 2026) in terms of retrieval performance, token usage, and runtime. Table 9 summarizes the results. Under **Average Time Taken**, we report the average indexing time per skill and retrieval time per query, both in seconds. Under **Token Usage**, we separately report the tokens consumed by LLM calls and those processed by embedding models, both in millions. We make two observations. From the tabel, we make two observations. 1) SkillGATE

Table 9: Comparison of efficiency between Structured SR methods.

	Ours	GoS	AgentSkillOS
Performance Metrics			
R@1	47.88 (100.00%)	28.07 (58.63%)	24.30 (50.75%)
R@10	73.83 (100.00%)	48.69 (65.95%)	29.66 (40.17%)
Token Usage (/M)			
LLM	56.08 (100.00%)	98.14 (175.00%)	848.21 (1512.50%)
Embedding	59.58 (100.00%)	38.85 (65.21%)	0.00 (0.00%)
Average Time Taken (/s)			
Index	0.15 (100.00%)	0.40 (266.67%)	0.81 (540.00%)
Retrieval	3.09 (100.00%)	0.63 (20.38%)	104.86 (3393.53%)

achieves the highest R@1/R@10 of 47.88/73.83 among the compared methods, using 56.08M LLM tokens and 59.58M embedding tokens. Notably, its LLM token consumption is 42.86% lower than that of GoS and 93.39% lower than that of AgentSkillOS. These results demonstrate that improved retrieval quality need not come at the expense of increased LLM token consumption: SkillGATE delivers stronger retrieval performance with substantially lower LLM token requirements. 2) The joint graph-tree index combines low construction overhead with strong retrieval performance. SkillGATE requires only 0.15 seconds per skill for indexing, compared with 0.40 seconds for GoS and 0.81 seconds for AgentSkillOS. For online retrieval, it achieves higher recall than AgentSkillOS while reducing the average time per query from 104.86 to 3.09 seconds, corresponding to a 33.9-fold speedup. Compared with GoS, it improves R@1/R@10 by 19.81/25.14 percentage points with an additional 2.46 seconds per query. Overall, SkillGATE achieves the lowest indexing time among the compared methods and substantially improves retrieval quality while keeping average retrieval latency at 3.09 seconds.

H CASE STUDY

Table 10: Case study comparing the retrieval traces of Vanilla MCTS and SKILLGATE.

Input Task

Query: What is the average speed of WN1507 from COS to MDW (miles per minute)?
Gold skill: Flight Data: Multi-Record Aggregation (`toolqa_003`).

Search Trace of Vanilla MCTS

#	Phase	Operation	Scope Transition	Size	Gold?
1	Selection	Node-aware descent	`root` → `p310`	1214	No
2	Expansion	Skill-aware descent	`p310` → `p387`	61	No
3	Simulation	Skill-aware descent	`p387` → `p22`	48	No
4	Simulation	Node-aware descent	`p22` → `p24`	17	No
5	Evaluation	Local candidate readouts	`p24` → `p24`	17	No

Search Trace of SkillGate

#	Phase	Operation	Scope Transition	Size	Gold?
1	Selection	Node-aware descent	`root` → `p310`	1,214	No
2	Expansion	Skill-aware descent	`p310` → `p387`	61	No
3	Simulation	Horizontal search	`p387` → `p1734`	17	No
4	Simulation	Horizontal search	`p1734` → `p1732`	21	**Yes**
5	Evaluation	Candidate collection	`p1732` → `p1732`	21	**Yes**

Examples of Skills in Visited Nodes

Node	Skill ID	Skill Title
`p310`	`web_07745`	Geographic Distance: Coordinate-Based Calculation
	`bigcodebench_000`	GeoPy: Distance Calculation and Geocoding
	`web_11953`	Astropy: Astronomical Data Analysis
`p387`	`web_07745`	Geographic Distance: Coordinate-Based Calculation
	`bigcodebench_000`	GeoPy: Distance Calculation and Geocoding
	`web_06171`	PostGIS: Spatial Queries and Routing
`p22`	`theoremqa_073`	Wave Mechanics: Speed, Frequency, and Wavelength
	`theoremqa_302`	Statistical Physics: Kinetic Theory and Fermi Statistics
	`theoremqa_016`	Physical Pendulum: Oscillation Period Calculation
`p24`	`theoremqa_073`	Wave Mechanics: Speed, Frequency, and Wavelength
	`theoremqa_302`	Statistical Physics: Kinetic Theory and Fermi Statistics
	`theoremqa_026`	Shock Waves: Mach Cone and Sonic Boom Arrival Time
`p1734`	`web_02816`	Google Maps: Place Search and Navigation
	`web_25548`	Google Maps CLI: Place Search and Route Planning
	`web_10840`	Location Search: Natural-Language Place Queries
`p1732`	`toolqa_002`	Flight Data: Single-Record Lookup
	`web_10302`	Route Planning: Distance, Duration, and Directions
	`toolqa_003`	**Flight Data: Multi-Record Aggregation**

I PROMPT TEMPLATES

Prompt Template for Structured Label Extraction

Role

You extract structured labels from executable agent skills for SkillGATE.

Grounding Constraints

Extract only information explicitly supported by the provided skill. Do not invent unsupported inputs, outputs, tools, APIs, or libraries. Use empty arrays when the corresponding information is unavailable.

Input / User Message (JSON)

```
{
"skill_id": "{skill_id}",
"name": "{skill_name}",
"description": "{skill_description}",
"content": "{skill_content}"
}
```

Response Format

Return JSON only with the following fields:

```
{
"inputs": [...],
"outputs": [...],
"tools_apis_libraries": [...]
}
```

Prompt Template for LLM-based Skill Reranking

Task

Given the following problem, rank the skills below by relevance.

Input / User Message

```
Problem:
{query}
Skills:
[1] {skill_name_1}: {skill_description_1}
[2] {skill_name_2}: {skill_description_2}
...
[K] {skill_name_K}: {skill_description_K}
```

Response Format

Output ONLY the skill numbers in order from most to least relevant, separated by commas.

```
Most relevant first (numbers only):
```