# Graph-of-Skills: Dependency-Aware Structural Retrieval for Massive Agent Skills

Dawei Liu*
University of Pennsylvania
Philadelphia, PA, USA
liudawei@seas.upenn.edu

Zongxia Li*
University of Maryland
College Park, USA
zli12321@umd.edu

Hongyang Du
Brown University
Providence, RI, USA

Xiyang Wu
University of Maryland
College Park, USA

Shihang Gui
Brown University
Providence, RI, USA

Yongbei Kuang
Carnegie Mellon University
Pittsburgh, PA, USA

Lichao Sun
Lehigh University
Bethlehem, PA, USA
lis221@lehigh.edu

## Abstract

Skill usage has become a core component of modern agent systems and can substantially improve agents' ability to complete complex tasks. In real-world settings, where agents must monitor and interact with numerous personal applications, web browsers, and other environment interfaces, skill libraries can scale to thousands of reusable skills. Scaling to larger skill sets introduces two key challenges. First, loading the full skill set saturates the context window, driving up token costs, hallucination, and latency. In this paper, we present **Graph-of-Skills (GoS)**, an inference-time structural retrieval layer for large skill libraries. GoS constructs an executable skill graph offline from skill packages, then at inference time retrieves a bounded, dependency-aware skill bundle through hybrid semantic–lexical seeding, reverse-weighted Personalized PageRank, and context-budgeted hydration. On SkillsBench and ALFWorld, GoS consistently delivers substantial reward improvements and token savings across three model families (Claude Sonnet 4.5, GPT-5.2 Codex, and MiniMax). On SkillsBench, GoS achieves a peak reward increase of 25.55% while reducing input tokens by 56.72% over the vanilla full skill-loading baseline using GPT-5.2 Codex. Additional ablation studies across skill libraries ranging from 200 to 2,000 skills further demonstrate that GoS consistently outperforms both vanilla skills loading and simple vector retrieval in balancing reward, token efficiency, and runtime. Code and skill artifacts: https://github.com/davidliuk/graph-of-skills.

***Keywords:*** Computer Use Agents, Agent Skills, Graph-of-Skills, Skill Retrieval

## 1 Introduction

Large Language Model (LLM) agents solve complex technical tasks by invoking external tools, APIs, and reusable skills [Mialon et al. 2023; Schick et al. 2023]. As these tools and skills grow from dozens of tools to thousands or even tens of thousands of candidates [Li et al. 2023; Patil et al. 2023; Qin et al. 2024; Xu et al. 2023], the core challenge shifts from deciding *whether* to use a skill to retrieving the most relevant set of skills that is sufficient for a task. Shi et al. [2025] already shows that skill retrieval itself is now a major bottleneck in realistic tool ecosystems.

Two common strategies are widely used for handling large skill libraries. **Vanilla Skills** [Agent Skills 2026] prepends the entire skill set to the context window. This can work for small toolsets, but it scales poorly: token cost grows linearly with library size, and critical domain constraints become easy for the model to overlook inside an overloaded context [Liu et al. 2024a]. An alternative is **vector-based retrieval** [Lewis et al. 2020; Wang et al. 2023], which improves efficiency by retrieving semantically similar skills. However, semantic proximity does not imply executable sufficiency. In many engineering tasks, the top semantic match is a high-level solver, while the actual solution also requires a lower-level parser, converter, setup utility, or domain-specific preprocessor that is semantically weak but functionally necessary [Patel et al. 2025; Patil et al. 2023; Qin et al. 2024] (Figure 1).

We present **Graph-of-Skills** (GoS), an inference-time structural retrieval layer for large local skill libraries to combat limitations of the previous two approaches. GoS constructs a directed multi-relational graph over local skill packages, where nodes are executable skills and edges encode prerequisite and workflow structure. At query time, GoS uses semantic and lexical signals only to identify a small seed set, then applies reverse-weighted Personalized PageRank

*Both authors contributed equally.

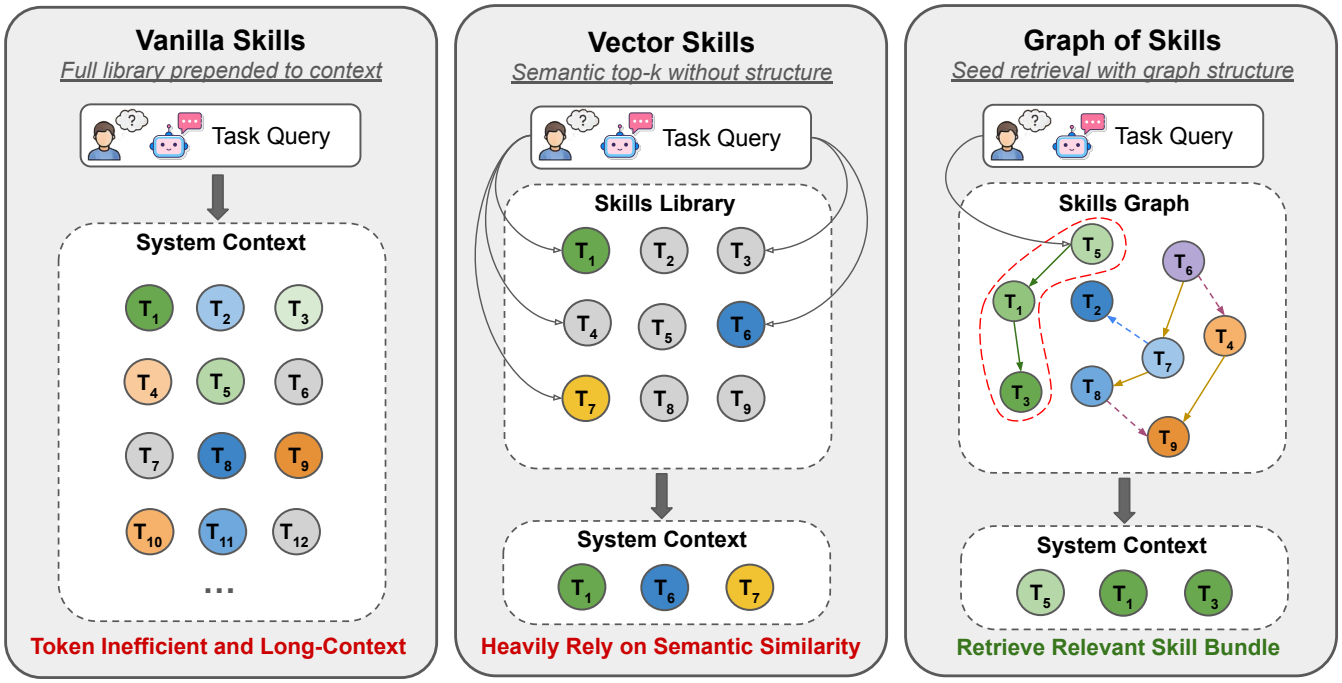



**Figure 1.** Conceptual comparison between flat skill loading, vector retrieval, and Graph-of-Skills (GoS). **Vanilla Skills** prepends the full skill library to the prompt, so relevant constraints and prerequisite skills become buried in an overloaded context. **Vector Skills** improves efficiency by returning semantically similar skills, but it can still miss a functionally required prerequisite outside the retrieved set, creating the prerequisite gap. **Graph-of-Skills** starts from hybrid semantic-lexical seeds and then performs structure-aware retrieval to recover prerequisite skills and assemble a compact execution bundle.

(PPR) [Haveliwala 2002; Yang et al. 2024] to recover additional skills that are structurally important for execution. The result is a bounded skill bundle that is both relevant and closer to dependency-complete than isolated top-$k$ retrieval. This problem setting is complementary to repository-scale skill infrastructures such as SkillNet and AgentSkillOS [Li et al. 2026c; Liang et al. 2026].

Those works focus on creating, organizing, evaluating, and orchestrating large skill ecosystems. GoS targets a critical downstream question: if a large local skill library already exists, how should an agent retrieve the smallest and most relevant executable subset that is sufficient for the current task? Rather than exposing only keyword or vector search over repository metadata, GoS parses each `skill specification` into executable fields such as I/O schemas, tooling, script entrypoints, and stable source paths, constructs typed dependency and workflow edges, and retrieves a bounded bundle through graph diffusion plus reranking.

Our contributions are as follows: (1) We introduce GoS, an agentic skill usage pipeline that combines offline graph construction with inference-time structural retrieval to improve skill selection accuracy while reducing input token consumption. (2) We evaluate GoS on the 1,000-skill SkillsBench setting and find that, compared to the full skill-loading baseline, GoS improves average reward by 43.6% and reduces input tokens by 37.8% across two benchmarks and three model families. Additional ablation studies confirm that this pattern holds consistently across skill library sizes ranging from 200 to 2,000 skills.

## 2 Related Work

**Tool Use, Tool Discovery, and Tool Retrieval for Agents.** Early research on tool-augmented language models focuses on relatively small, fixed toolsets, where the primary challenge is deciding *when* to invoke a tool and formatting the call correctly [Mialon et al. 2023; Schick et al. 2023]. As tool sets grow from dozens of tools to thousands [Li et al. 2023; Patil et al. 2023; Qin et al. 2024; Xu et al. 2023] and context windows continue to expand [Comanici et al. 2025; Li et al. 2026b; Singh et al. 2026], the problem shifts toward *tool discovery* and *tool retrieval*. Systems and benchmarks such as Gorilla [Patil et al. 2023], API-Bank [Li et al. 2023], ToolBench-style evaluations [Xu et al. 2023], and ToolLLM [Qin et al. 2024] show that large tool universes require scalable retrieval over API descriptions and tool documentation. ToolNet [Liu et al. 2024b] introduces graph structure into large-scale tool access, with the objective to connect models to broad tool ecosystems rather than recover dependency-complete local executable bundles. However, Shi et al. [2025] shows that tool retrieval is itself a difficult modeling problem and that

generic dense retrievers are often poorly aligned with real tool-use needs [Shi et al. 2025].

**Skill Repositories, Ecosystems, and Benchmarks.** Recent systems increasingly treat agent skills as reusable assets rather than ad hoc prompts [Agent Skills 2026; Li et al. 2026c; Liang et al. 2026]. SkillNet [Liang et al. 2026] is a repository-scale skill library in this space, supporting skill creation from heterogeneous sources, multi-dimensional evaluation, ontology construction, and relational analysis over large skill collections. AgentSkillOS [Li et al. 2026c] similarly advocates for an ecosystem-level approach, emphasizing that massive skill libraries must be systematically categorized for efficient retrieval and dynamically chained together to execute complex, multi-step tasks. SkillsBench [Li et al. 2026a] shows that curated external skills can improve agent performance, while also exposing that simply having many skills available does not guarantee reliable and safe use. Other systems and registries such as SkillsMP, ClawHub, and LangSkills similarly support packaging, discovery, and search over large skill collections [LabRAI 2026; Li et al. 2025; OpenClaw 2026; SkillsMP 2026]. These skill repository platforms lower the cost of packaging, publishing, browsing, and searching large skill collections, but their primary interface remains entry-level search or distribution over individual skills or bundles.

**Graph-Based Retrieval and Relational Memory.** Graph-structured retrieval has recently improved knowledge access in document, memory, and tool-use settings, but its role differs substantially across these regimes. GraphRAG [Edge et al. 2024] uses graph structure to support query-focused synthesis over document collections, HippoRAG [Jiménez Gutiérrez et al. 2024] models long-term memory as an associative graph for improved retrieval, and adjacent agent systems such as ControlLLM [Liu et al. 2023] and ToolNet [Liu et al. 2024b] incorporate graph structure over tools rather than treating tools as a flat list. However, these lines of work do not directly study retrieval over large local skill repositories. GraphRAG-style systems target knowledge synthesis, memory access, or relational QA; tool-graph methods focus primarily on graph-guided tool planning and navigation during reasoning. By contrast, our setting requires an *upstream* retrieval layer that selects a small executable bundle *before* generation begins. To our knowledge, prior work has not focused on graph-based retrieval for agent skills under this objective: recovering a dependency-complete executable bundle under a tight context budget, rather than merely retrieving one relevant item.

# 3 Methodology

GoS is an inference-time retrieval layer for large local skill libraries. It constructs a typed graph offline from local skill packages and, at query time, returns a compact execution bundle that is relevant to the task and more likely than flat retrieval to include the prerequisites required for successful execution.

## 3.1 Problem Setup

Let $C = \{d_1, \ldots, d_m\}$ denote a local corpus of skill packages. Each package contains a primary specification document together with optional scripts, references, and auxiliary assets. GoS converts $C$ into a typed directed graph

$$G = (V, E, w, \phi),$$

where each node $v \in V$ is a normalized executable skill record, each edge $e \in E$ connects two skills, $w(e) > 0$ is an edge weight, and $\phi(e) \in \mathcal{R}$ assigns an edge type from the relation set

$$\mathcal{R} = \{\text{dep}, \text{wf}, \text{sem}, \text{alt}\}.$$

Given a task query $q$ and a context budget $\tau$, the retrieval problem is to return a bundle $B(q) \subseteq V$ that is simultaneously relevant, execution-complete when possible, and compact. We view this as a budgeted selection problem,

$$\max_{B \subseteq V} \sum_{v \in B} \text{rel}(v, q) + \beta \sum_{(u,v) \in E_{\text{dep}}} \mathbb{I}[u \in B \wedge v \in B] \quad \text{s.t. } \text{cost}(B) \leq \tau, \tag{1}$$

where the first term favors query relevance, the second rewards dependency-complete bundles, and $\text{cost}(B)$ measures the prompt budget consumed by the hydrated bundle. Equation (1) is not solved exactly. Instead, GoS approximates this objective through three stages: hybrid seed retrieval, reverse-aware graph diffusion, and budgeted reranking plus hydration.

## 3.2 Offline Graph Construction

**Skill Normalization.** Each skill package is parsed into a normalized skill record containing a canonical name, capability summary, I/O fields, domain tags, tooling, entrypoints, compatibility notes, and a stable local source path. This normalization step is primarily deterministic: the system extracts executable fields whenever possible and uses a lightweight LLM pass only to recover retrieval-critical semantic fields when package documentation is incomplete. The goal is to convert each local skill into a retrieval unit that an agent can directly consume at inference time.

**Typed Relation Induction.** GoS uses four edge types. *Dependency* edges represent executable prerequisites and form the primary structural relation in the graph. A dependency edge $v_i \rightarrow v_j$ is added *deterministically* whenever the I/O schemas of $v_i$ and $v_j$ overlap above a fixed threshold, so that $v_i$ can plausibly provide an artifact consumed by $v_j$; no LLM call is involved. *Workflow* edges capture common multi-step pipelines, *semantic* edges connect near-duplicate or topically adjacent skills, and *alternative* edges link interchangeable strategies for the same subproblem.

Rather than performing unconstrained all-pairs relation inference, GoS constructs the three non-dependency edge

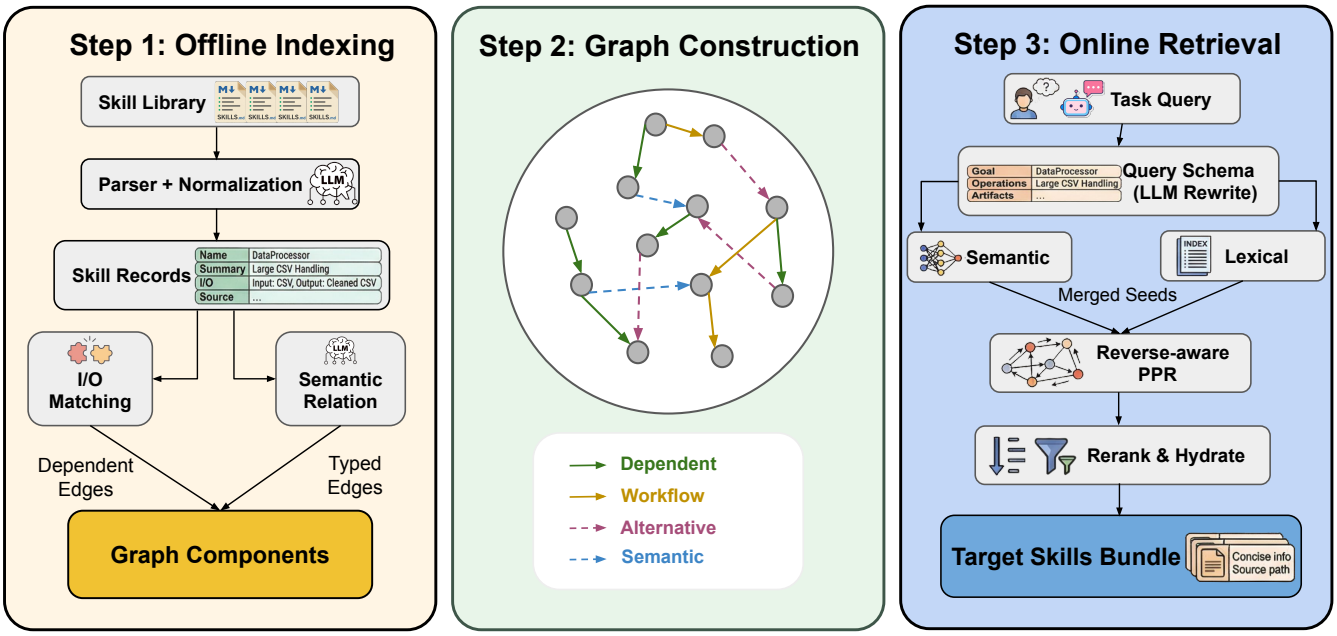



**Figure 2.** Overview of Graph of Skills (GoS). **Left:** offline indexing converts local skill packages into normalized skill records and typed edges. Dependency edges are induced from I/O compatibility, while workflow, semantic, and alternative relations are added through sparse validation. **Center:** the typed directed skill graph is the retrieval substrate; edge labels denote dependency, workflow, semantic, and alternative relations. **Right:** online retrieval maps a task query to a compact query schema, forms merged seeds from semantic and lexical retrieval, applies reverse-aware Personalized PageRank, and returns a budgeted execution bundle after reranking and hydration.

types through sparse LLM validation. For each node, it first forms a small candidate pool using lexical similarity, semantic neighbors, and I/O-based expansion, and only the top-$k$ candidates per node are submitted to a constrained validator prompt (default $k$=8). Total LLM validation calls therefore scale as $O(Nk)$ rather than $O(N^2)$, and incremental indexing further reduces this to $O(|V_{\text{new}}|\, N)$. This keeps graph construction tractable while anchoring the resulting graph in executable structure rather than metadata proximity alone.

### 3.3 Online Structural Retrieval

**Query Representation and Hybrid Seeding.** Dense retrieval [Karpukhin et al. 2020] is often effective at finding the visible top-level skill but weak at recovering semantically subtle prerequisites. Lexical retrieval in the probabilistic ranking tradition [Robertson et al. 2009] is robust for concrete artifacts and filenames, but brittle under paraphrase. GoS therefore combines the two signals at the seeding stage.

At retrieval time, the raw query is mapped to a lightweight retrieval schema containing the task goal, salient operations, referenced artifacts, and normalized keywords. This schema can be produced by an optional LLM rewrite, following the general intuition of rewrite-then-retrieve pipelines [Ma et al. 2023]; when rewriting is unavailable or disabled, GoS falls back to deterministic lexical normalization. GoS then computes a semantic seed score $s_i^{\text{sem}}(q)$ and a lexical seed score $s_i^{\text{lex}}(q)$ for each candidate skill $v_i$, and merges them as

$$z_i(q) = \eta\, s_i^{\text{sem}}(q) + (1-\eta)\, s_i^{\text{lex}}(q), \tag{2}$$

where $\eta \in [0,1]$ controls the semantic–lexical tradeoff. The initial seed distribution is obtained by normalizing the merged scores over the candidate pool,

$$\mathbf{p}_i = \frac{z_i(q)}{\sum_j z_j(q)}.$$

**Reverse-Aware Typed Diffusion.** Let $A_r$ denote the weighted adjacency matrix for relation type $r \in \mathcal{R}$. To let retrieval move from a matched high-level skill toward likely prerequisites, GoS uses both forward and reverse transitions. For each relation type, we define a row-normalized forward operator $T_r^{\rightarrow}$ and a row-normalized reverse operator $T_r^{\leftarrow}$ obtained from $A_r$ and $A_r^{\top}$, respectively. GoS then forms the unified transition operator

$$T = \text{RowNorm}\left(\sum_{r\in\mathcal{R}} \lambda_r \left(T_r^{\rightarrow} + \gamma_r T_r^{\leftarrow}\right)\right), \tag{3}$$

where $\lambda_r \geq 0$ and $\sum_r \lambda_r = 1$ weight relation types, and $\gamma_r \geq 0$ controls how strongly reverse traversal is allowed for each type. In practice, reverse edges are inserted directly

into the transition matrix with type-specific weights (largest for dependency, smallest for alternative; concrete values in Appendix 6), and the matrix is row-normalized before diffusion.

The core retrieval step is a reverse-aware Personalized PageRank-style diffusion over this operator [Jeh and Widom 2003; Page et al. 1999; Yang et al. 2024]:

$$\mathbf{s}^{(\ell+1)} = \alpha\,\mathbf{p} + (1-\alpha)T^{\top}\mathbf{s}^{(\ell)}, \tag{4}$$

where $\alpha \in (0,1)$ is the restart parameter. Relative to flat top-$k$ retrieval, relevance is not assigned only to individually matched skills; it is propagated across a local executable neighborhood. In particular, once a high-level solver is retrieved as a seed, upstream parser, setup, or preprocessing skills can still accumulate score through reverse dependency paths even when they are not themselves strong semantic matches to the original query.

**Budgeted Reranking and Hydration.** The diffusion score alone is insufficient, because the final output must be compact and directly usable by an agent. GoS therefore reranks candidate skills by combining graph score with field-level query evidence:

$$\rho_i(q) = \mathbf{s}_i^{\star} + \mu\, m_i(q), \tag{5}$$

where $\mathbf{s}_i^{\star}$ is the converged diffusion score, $m_i(q)$ aggregates direct matches between the query and skill fields such as name, capability summary, artifacts, and entrypoints, and $\mu$ controls how much local grounding is preserved after graph expansion.

Candidates are then hydrated in descending order of $\rho_i(q)$ under both per-skill and global context budgets. Here, *hydration* denotes materializing a selected skill into an agent-consumable payload that includes a stable source path together with concise capability text and the most relevant execution notes. The final output is therefore a bounded execution bundle designed to maximize executable coverage within the prompt budget.

## 4 Experiments

We evaluate whether graph-structured retrieval improves agent performance and efficiency relative to flat full-library access and non-graph semantic retrieval.

### 4.1 Experimental Setup

We evaluate GoS on two benchmarks using the full released task sets. **SkillsBench** [Li et al. 2026a] contains a diverse set of real-world technical tasks across 11 domains, paired with curated Skills: structured packages of procedural knowledge (instructions, code templates, resources) that augment LLM agents at inference time. The task domains span complex technical work such as macroeconomic detrending, power-grid feasibility analysis, 3D scan analysis, financial modeling, and seismic phase picking. **ALFWorld** [Shridhar et al. 2020b] is an interactive simulator that aligns text descriptions and commands with a physically embodied robotic environment, built by combining TextWorld [Côté et al. 2018], an engine for interactive text-based games and the ALFRED dataset [Shridhar et al. 2020a]. Its tasks involve multi-step household activities such as navigating rooms, finding objects, and manipulating them. In the LLM agent literature, ALFWorld is widely used in its text-only mode as a benchmark for sequential decision making, where an agent receives textual room descriptions and must issue a chain of commands to accomplish a goal [Shinn et al. 2023; Yao et al. 2023]. We evaluate on the full 140-episode split.

**Baselines.** We compare GoS against two baselines. *Vanilla Skills* exposes the entire skill library directly to the agent, maximizing recall but providing no retrieval-time compression. On ALFWorld, this follows the official Agent Skills format and reference repository [Agent Skills 2026]. *Vector Skills* retrieves a bounded set of skills using semantic similarity over the same embedding model used by GoS, namely `openai/text-embedding-3-large` [OpenAI 2024] (3072 dimensions). It isolates the effect of graph structure from the general benefit of retrieval-time compression. *GoS* uses the same base embedding model as *Vector Skills* but replaces flat nearest-neighbor retrieval with structure-aware retrieval over the skill graph. We disable the optional query-rewrite module so that any observed gain can be attributed to graph propagation rather than to an LLM rewrite of the query, and use the raw task instruction as the retrieval query. The critical comparison is therefore between flat semantic retrieval and dependency-aware structural retrieval under the same backbone and embedding setup.

**Models and Evaluation.** All experiments are conducted with Claude Sonnet 4.5 [Anthropic 2025], MiniMax M2.7 [MiniMax 2026], and GPT-5.2 Codex [OpenAI 2025]. Each model–method setting is run twice, and we report the mean across runs. We report average reward across tasks as the primary evaluation metric. For ALFWorld, rewards are binary, so average reward is equivalent to success rate. We additionally report average total token usage and agent-only runtime; runtime is measured from agent start to agent finish and excludes environment setup.

**Evaluation Protocol.** We use the full benchmark task sets and apply the same retry policy across the main and sensitivity experiments. If environment construction fails, we rebuild and rerun the task up to two additional times; tasks that still fail after these retries are excluded as unresolved infrastructure failures rather than counted as model failures. For agent timeouts, we distinguish between substantive execution failures and startup failures: if the agent has already been executing for a long time and then times out, we record reward 0 and keep the run in the aggregate; if the timeout occurs before a meaningful run is established, we rerun the trial. This protocol is applied to the full SkillsBench evaluation, the full 140-episode ALFWorld evaluation, and the library-size sensitivity study.

### 4.2 Main Results

We present the main results in Table 1. Across all six model–benchmark blocks, GoS attains the highest average reward. Relative to *Vanilla Skills*, it reduces average token usage in all six blocks and reduces agent runtime in five of the six. Relative to *Vector Skills*, it improves reward in every block while keeping the token budget in the same compressed regime.

**Naive semantic retrieval struggles on long-horizon tasks.** Many tasks in SkillsBench are long-horizon and require combining relevant skills with prerequisite utilities, such as environment setup, data preprocessing, or output formatting. These skills may not be lexically salient in the task description. Vector Skills, which retrieves based solely on embedding similarity to the query, often misses these indirect but essential dependencies, leading to incomplete skill sets and lower task completion rates. This pattern is most visible on SkillsBench. Under Claude Sonnet 4.5, *Vector Skills* drops from 25.0 to 19.3 average reward relative to *Vanilla Skills*; under MiniMax M2.7, it drops from 17.2 to 10.4; and under GPT-5.2 Codex, it drops from 27.4 to 21.5. In contrast, GoS improves over both baselines in all three SkillsBench blocks while still using substantially fewer tokens than flat prompting. These results are consistent with the hypothesis that long-horizon tasks are sensitive not only to topical relevance, but also to whether the retrieved bundle contains the prerequisite helpers needed to complete the full execution path.

**GoS achieves the strongest overall tradeoff on ALFWorld.** The ALFWorld results show that the same advantage transfers to a sequential embodied environment. Under Claude Sonnet 4.5, GoS reaches 97.9% average success, compared with 93.6% for *Vector Skills* and 89.3% for *Vanilla Skills*, while reducing average total tokens from 1,524,401 to 27,215 relative to flat prompting. Under MiniMax M2.7, GoS again gives the strongest overall tradeoff, improving reward from 47.1% under *Vanilla Skills* and 50.7% under *Vector Skills* to 54.3%, while also achieving the lowest token usage and runtime in that block. Under GPT-5.2 Codex, GoS and *Vector Skills* are close on reward (93.6% vs. 92.9%), but GoS still remains clearly more efficient than *Vanilla Skills*. Taken together, these results suggest that the benefit of structure-aware retrieval is not limited to technical code-execution tasks.

**GoS offers the best efficiency–performance tradeoff.** *Vanilla Skills* preserves maximal recall, but its cost grows rapidly with library size and leaves the agent to search an unstructured skill set at inference time. *Vector Skills* reduces token cost, but its retrieved set is often incomplete, because semantically nearby skills are not always jointly sufficient. GoS improves reward over *Vector Skills* by 10.97 points on SkillsBench and 2.87 points on ALFWorld while remaining far more efficient than *Vanilla Skills* (Table 1). The results are averaged over two runs per setting. We interpret them as a consistent empirical pattern rather than a formal significance claim.

### 4.3 Qualitative Analysis

We inspect trajectory-level evidence to study how retrieval quality changes the downstream execution path, not just the final score. Appendix F provides a broader case set; here we focus on one representative example that isolates the main mechanism behind GoS.

**`Pedestrian Traffic Counting.`** `pedestrian-traffic-counting` requires a short but complete visual pipeline: extracting frames, counting pedestrians reliably, and formatting the output in the expected structure. In this case, GoS retrieved a compact bundle centered on `gemini-count-in-video`, `video-frame-extraction`, and `openai-vision`, and achieved the highest score (0.417). *Vanilla Skills* eventually opened related helpers, including `gemini-count-in-video`, `video-frame-extraction`, and `object_counter`, but reached only 0.267, suggesting that broad library access can recover relevant tools while still leaving the agent with a noisier search problem. *Vector Skills* scored only 0.041: it retrieved some relevant context, but not a bundle that the agent could convert into a workable end-to-end plan. The qualitative lesson is that GoS does not help merely by retrieving topically similar skills. It helps by exposing a bundle that is already close to the executable decomposition of the task, so the agent can commit earlier to a verifier-aligned plan rather than spending budget on additional search and assembly.

## 5 Ablation Study

We conduct an additional ablation study to evaluate the impact of the skill library size on GoS, *Vanilla Skills*, and *Vector Skills*, alongside the effects of the lexical merge and reranker components on GoS performance.

### 5.1 Sensitivity to Skill Library Size

We run an additional full-SkillsBench study with GPT-5.2 Codex while varying the library size from 200 to 500, 1,000, and 2,000 skills. We report average reward, average input tokens, and agent-only runtime under the same retry and exclusion rules used in the main experiments in Table 2.

**Prompt cost grows rapidly for all skill exposure.** The strongest trend is on input tokens. As the library grows from 500 to 2,000 skills, *Vanilla Skills* rises from 1.93M to 5.84M average input tokens, roughly a 3× increase. Over the same range, *Vector Skills* stays near 1.10M–1.24M tokens and GoS stays near 1.14M–1.38M tokens. This result shows that simple retrieval substantially weakens the coupling between repository size and prompt size, while GoS does so without giving up reward.

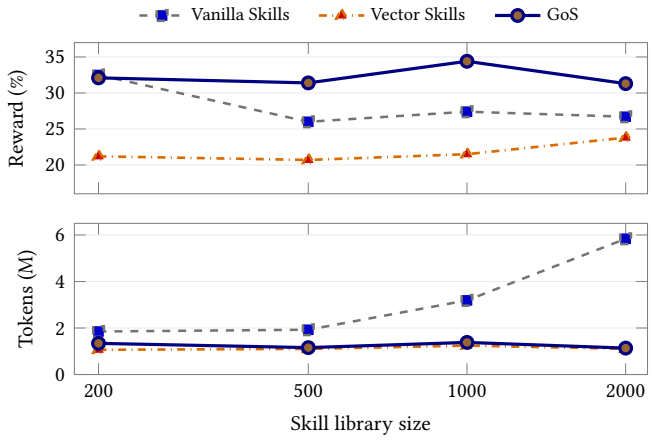
**Table 1. R** denotes average reward (%), **T** denotes tokens, and **S** denotes runtime (**s**) (↑ indicates larger values are better, and ↓ denotes smaller values are better). Results are means over two runs per setting. For ALFWorld, average reward equals success rate. The top-performing results are highlighted in **bold**, and the second-best are underlined.

| | | SkillsBench | | | ALFWorld | | |
|---|---|---|---|---|---|---|---|
| Model | Method | R ↑ | T ↓ | S ↓ | R ↑ | T ↓ | S ↓ |
| Claude Sonnet 4.5 | Vanilla Skills | 25.0 | 967,791 | 465.8 | 89.3 | 1,524,401 | 53.2 |
| | Vector Skills | 19.3 | 894,640 | **357.3** | 93.6 | 28,407 | **37.8** |
| | + GoS | **31.0** | **860,315** | 364.9 | **97.9** | **27,215** | 49.2 |
| MiniMax M2.7 | Vanilla Skills | 17.2 | 942,113 | 580.7 | 47.1 | 2,184,823 | 88.6 |
| | Vector Skills | 10.4 | **852,881** | 552.9 | 50.7 | 66,109 | 73.4 |
| | + GoS | **18.7** | 867,452 | **502.5** | **54.3** | **65,227** | **68.8** |
| GPT-5.2 Codex | Vanilla Skills | 27.4 | 3,187,749 | **686.8** | 89.3 | 1,435,614 | 83.3 |
| | Vector Skills | 21.5 | **1,243,648** | 773.0 | 92.9 | **34,436** | **57.0** |
| | + GoS | **34.4** | 1,379,773 | 715.6 | **93.6** | 46,462 | 64.7 |

**Figure 3.** Sensitivity to library size on full SkillsBench under GPT-5.2 Codex. Left: compact summary table for Vanilla Skills, Vector Skills, and GoS. Right: reward and input-token trends as the skill repository grows from 200 to 2,000 skills. GoS preserves the strongest reward once the library becomes moderately large, while both retrieval-based methods substantially weaken the growth of prompt cost relative to flat exposure.

| N | Method | R ↑ | T ↓ | S ↓ |
|---|---|---|---|---|
| 200 | Vanilla Skills | **32.5** | 1.85 | **701.6** |
| | Vector Skills | 21.2 | **1.06** | 833.8 |
| | + GoS | 32.1 | 1.36 | 731.2 |
| 500 | Vanilla Skills | 26.0 | 1.93 | **756.8** |
| | Vector Skills | 20.7 | **1.10** | 849.5 |
| | + GoS | **31.4** | **1.16** | 890.3 |
| 1000 | Vanilla Skills | 27.4 | 3.19 | **686.8** |
| | Vector Skills | 21.5 | **1.24** | 773.0 |
| | + GoS | **34.4** | 1.38 | 715.6 |
| 2000 | Vanilla Skills | 26.7 | 5.84 | **733.5** |
| | Vector Skills | 23.8 | **1.11** | 799.8 |
| | + GoS | **31.3** | 1.14 | 788.0 |

**N**: library size, **R**: reward, **T**: input tokens (M), **S**: runtime (s).




**GoS maintains a reward advantage at all tested scales.** At 200 skills, *Vanilla Skills* is still slightly ahead of GoS (32.5 vs. 32.1). Once the library becomes moderately large, GoS outperforms both baselines at every tested scale: 31.4 vs. 26.0 / 20.7 at 500 skills, 34.4 vs. 27.4 / 21.5 at 1,000 skills, and 31.3 vs. 26.7 / 23.8 at 2,000 skills (GoS / Vanilla Skills / Vector Skills). The margin is largest at 1,000 skills and remains substantial at 2,000, indicating that increasing library size does not weaken the benefit of dependency-aware retrieval. **The extra retrieval step adds modest runtime for GPT-5.2 Codex, but does not change the main scaling conclusion.** Both retrieval-based methods are slower than *Vanilla Skills* in agent-only runtime for GPT at most scales, reflecting the overhead of searching before execution. However, this reduced runtime is unique to GPT-5.2 Codex, likely due to caching mechanisms for fixed skill libraries within the black-box model, where Claude and MiniMax have longer runtime when using *Vanilla Skills* than GoS and *Vector Skills* (Table 1). In contrast, the Claude model lacks this optimization, making the *Vanilla Skills* approach significantly slower than retrieval methods. Furthermore, the results suggest that the primary system bottleneck is not graph traversal or vector search, but rather the overhead of exposing an increasingly large, flat library directly to the model.

### 5.2 Component Analysis of the GoS Retrieval Pipeline

We next evaluate the contribution of two key retrieval components in the GoS pipeline: graph propagation and lexical reranking. These ablations were run under the main SkillsBench configuration — GPT-5.2 Codex across skill libraries

| Method | R ↑ | T ↓ | S ↓ |
|---|---|---|---|
| Full GoS | **34.4** | 1.38 | 715.6 |
| w/o graph propagation | 29.3 | 0.89 | 766.2 |
| w/o lexical + rerank | 26.7 | 1.01 | 747.7 |

**Table 2.** Component ablation on full SkillsBench with GPT-5.2 Codex and the 1,000-skill library. **R**: average reward (%), **T**: average total tokens (M), **S**: agent-only runtime (s).

of increasing size (200, 500, 1,000, and 2,000 skills). In the first ablation, we remove graph propagation, disabling the system's ability to expand beyond seed skills to structurally related prerequisites. In the second, we remove lexical retrieval and reranking, forcing the system to rely solely on the semantic retriever before graph expansion.

**Graph propagation and lexical reranking are important components for GoS' success.** Removing graph propagation reduces average token usage from 1.38M to 0.89M, but it also lowers average reward from 34.4 to 29.3 (↓ 5.1). Removing lexical retrieval and reranking lowers average token usage from 1.38M to 1.01M. It lowers average reward from 34.4 to 26.7 (↓ 7.7). The larger degradation in the second ablation suggests that better seed quality is especially important on SkillsBench: if the initial retrieved skills are weak, graph expansion has less useful structure from which to recover missing prerequisites. These results show that hybrid semantic–lexical retrieval improves entry-point quality, and graph propagation then converts those stronger seeds into a more execution-complete bundle.

## 6 Conclusion

Skill retrieval is a critical bottleneck for agents operating over massive skill libraries. Unlike approaches that retrieve only semantically relevant skills, GoS recovers a small, jointly sufficient set of skills by capturing not just the target skill but also the parsers, preprocessors, and dependencies needed for successful execution. GoS is complementary to, but distinct from, broader skill management systems such as SkillNet and AgentSkillOS [Li et al. 2026c; Liang et al. 2026]. Our results demonstrate that GoS consistently outperforms both vanilla skills loading and simple vector retrieval, improving execution reward while reducing token consumption. This advantage holds across two benchmarks, three model families, and skill libraries of varying sizes.

**Limitations and Future Work.** GoS still depends on the quality of its offline graph. Poorly documented skills, ambiguous I/O schemas, or missing execution metadata can degrade edge quality and downstream retrieval. In addition, the current graph system is mostly static: it does not yet refine graph structure from repeated execution traces, verifier outcomes, or user feedback. Future work can focus on including online edge-weight adaptation, graph updates from successful trajectories, stronger reranking over candidate bundles, and broader evaluation on multimodal and interactive agent settings.

## References


Agent Skills. 2026. Agent Skills. https://github.com/agentskills/agentskills Specification and documentation repository, accessed 2026-04-01.

Anthropic. 2025. Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5 Official release page, accessed 2026-04-01.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2018. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*. Springer, 41–75.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130* (2024). arXiv:2404.16130

Taher H. Haveliwala. 2002. Topic-sensitive PageRank. In *Proceedings of the 11th International Conference on World Wide Web* (Honolulu, Hawaii, USA) *(WWW '02)*. Association for Computing Machinery, New York, NY, USA, 517–526. doi:10.1145/511446.511513

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

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831* (2024).

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6769–6781. doi:10.18653/v1/2020.emnlp-main.550

LabRAI. 2026. LangSkills. https://github.com/LabRAI/LangSkills GitHub repository, accessed 2026-04-01.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026c. Organizing, Orchestrating, and Benchmarking Agent Skills at Ecosystem Scale. *arXiv preprint arXiv:2603.02176* (2026). arXiv:2603.02176

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. *arXiv preprint arXiv:2304.08244* (2023). arXiv:2304.08244

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-chung Lee. 2026a. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670* (2026). arXiv:2602.12670 doi:10.48550/arXiv.2602.12670

Zongxia Li, Hongyang Du, Chengsong Huang, Xiyang Wu, Lantao Yu, Yicheng He, Jing Xie, Xiaomin Wu, Zhichao Liu, Jiarui Zhang, et al. 2026b. Mm-zero: Self-evolving multi-model vision language models from zero data. *arXiv preprint arXiv:2603.09206* (2026).

Zongxia Li, Wenhao Yu, Chengsong Huang, Rui Liu, Zhenwen Liang, Fuxiao Liu, Jingxi Che, Dian Yu, Jordan Boyd-Graber, Haitao Mi, et al. 2025. Self-rewarding vision-language model via reasoning decomposition. *arXiv preprint arXiv:2508.19652* (2025).

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. 2026. SkillNet: Create, Evaluate, and Connect AI Skills. *arXiv preprint arXiv:2603.04448* (2026). arXiv:2603.04448

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. doi:10.1162/tacl_a_00638

Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. 2024b. ToolNet: Connecting Large Language Models with Massive Tools via Tool Graph. *arXiv preprint arXiv:2403.00839* (2024). arXiv:2403.00839

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2023. ControlLLM: Augment Language Models with Tools by Searching on Graphs. *arXiv preprint arXiv:2310.17796* (2023). arXiv:2310.17796

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query Rewriting in Retrieval-Augmented Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 5303–5315. doi:10.18653/v1/2023.emnlp-main.322

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *Transactions on Machine Learning Research* (2023).

MiniMax. 2026. MiniMax M2.7. https://www.minimax.io/news/minimax-m27-en Official release page, accessed 2026-04-01.

OpenAI. 2024. text-embedding-3-large. https://developers.openai.com/api/docs/models/text-embedding-3-large Official model documentation, accessed 2026-04-01.

OpenAI. 2025. GPT-5.2-Codex. https://developers.openai.com/api/docs/models/gpt-5.2-codex Official model documentation, accessed 2026-04-01.

OpenClaw. 2026. ClawHub Registry. https://openclawdoc.com/docs/skills/clawhub/ Documentation page, accessed 2026-04-01.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

Bhrij Patel, Davide Belli, Amir Jalalirad, Maximilian Arnold, Aleksandr Ermovol, and Bence Major. 2025. Dynamic Tool Dependency Retrieval for Efficient Function Calling. *arXiv preprint arXiv:2512.17052* (2025). arXiv:2512.17052

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334* (2023). arXiv:2305.15334

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations*. arXiv:2307.16789

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *Advances in Neural Information Processing Systems* 36 (2023). arXiv:2302.04761

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval Models Aren't Tool-Savvy: Benchmarking Tool Retrieval for Large Language Models. *arXiv preprint arXiv:2503.01763* (2025). arXiv:2503.01763

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366* (2023). arXiv:2303.11366

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10740–10749.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020b. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768* (2020).

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. 2026. OpenAI GPT-5 System Card. *arXiv preprint arXiv:2601.03267* (2026). arXiv:2601.03267

SkillsMP. 2026. SkillsMP. https://skillsmp.com/ Agent Skills Marketplace, accessed 2026-04-01.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291* (2023). arXiv:2305.16291

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the Tool Manipulation Capability of Open-source Large Language Models. *arXiv preprint arXiv:2305.16504* (2023). arXiv:2305.16504

Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. 2024. Efficient Algorithms for Personalized PageRank Computation: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 9 (2024), 4582–4602. doi:10.1109/TKDE.2024.3376000

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*.

# A Appendix Overview

To make the supplementary material easier to navigate, we briefly summarize the organization of the appendix before presenting the detailed evidence. The appendix is designed to complement the main paper along four axes: implementation fidelity, prompt/interface design, retrieval mechanics, and trajectory-grounded empirical analysis.

**Table 3.** Appendix roadmap. The table summarizes the role of each supplementary section and how it complements the main paper.

| Section | Purpose |
|---|---|
| Implementation Details | Documents how GoS is instantiated in code, including parsing, graph construction, hybrid seeding, diffusion, reranking, and hydration. |
| Prompt and Interface Examples | Shows representative internal prompts and agent-facing interface rules used during graph construction and online retrieval. |
| Core Retrieval Pseudocode | Gives compact pseudocode for the offline indexing pipeline and the online graph-based retrieval pipeline. |
| Error Analysis | Separates retrieval misses, partial retrieval, execution drift, and infrastructure failures. |
| Qualitative Analysis | Provides trajectory-grounded case studies that compare the actual skill context exposed under different retrieval conditions. |

# B Implementation Details

This appendix summarizes the implementation decisions behind GoS and clarifies how the abstract pipeline in the main text is instantiated in code. The purpose is not to enumerate every engineering detail, but to expose the concrete design choices that determine graph quality, retrieval behavior, and the final agent-facing bundle.

**Table 4.** GoS implementation summary. The table highlights the main design choices that determine graph construction, retrieval, and agent-facing hydration.

| Component | Implementation choice |
|---|---|
| Node construction | Parser-first normalization from `SKILL.md` plus optional LLM completion of retrieval-critical semantic fields. |
| Dependency edges | Directed edges induced by bidirectional output–input compatibility checks. |
| Higher-order edges | Sparse LLM validation over a bounded candidate pool for workflow, semantic, and alternative relations. |
| Seed retrieval | Hybrid semantic and lexical seeding over normalized node fields. |
| Graph scoring | Reverse-aware typed Personalized PageRank with relation-specific reverse transitions. |
| Final output | Reranked, budgeted hydration into agent-usable skill payloads with stable `Source:` paths. |

**Table 5.** Normalized node fields used at retrieval time. The table lists the fields retained in each skill node and their retrieval role in GoS.

| Field | Primary role | Why it matters |
|---|---|---|
| `name`, `description` | canonical identity and coarse semantic match | Provide the most stable high-level skill signature during lexical and semantic seeding. |
| `one_line_capability` | concise capability abstraction | Helps retrieval align a task to what the skill actually does, rather than to document wording alone. |
| `inputs`, `outputs` | executable interface schema | Support deterministic dependency induction and help retrieval recover prerequisite producers and consumers. |
| `domain_tags`, `tooling` | technical context | Improve matching on domain-specific libraries, APIs, and workflows when task language is underspecified. |
| `example_tasks` | usage priors | Improve recall for tasks described by objective or scenario rather than by direct tool names. |
| `script_entrypoints` | implementation affordances | Help the agent discover reusable scripts instead of reimplementing logic from scratch. |
| `compatibility`, `allowed_tools` | execution constraints | Preserve operational restrictions that are important for verifier-aligned use. |
| `source_path`, `rendered_snippet` | hydration and agent consumption | Make retrieved skills directly inspectable inside the execution environment and keep the bundle compact. |

**Table 6.** Relation types and edge weights used in reverse-aware graph diffusion. Forward weights govern transition mass along the stored edge direction; reverse weights govern backward propagation from a matched skill toward its likely prerequisites. Dependency edges, the only deterministically induced relation, receive the largest weight in both directions; the three LLM-validated relations receive smaller reverse weights to limit topical drift during diffusion.

| Relation | Forward | Reverse | Meaning | Retrieval consequence |
|---|---|---|---|---|
| Dependency | 1.0 | 1.0 | Skill $u$ produces an artifact consumed by skill $v$ | Strongest forward and backward propagation, since recovering prerequisites is the main purpose of GoS. |
| Workflow | 0.7 | 0.5 | Two skills are commonly chained in a concrete multi-step pipeline | Allows moderate backward expansion toward adjacent pipeline stages without dominating dependency evidence. |
| Semantic | 0.4 | 0.2 | Two skills belong to the same narrow capability cluster | Provides weak smoothing across near-neighbor skills while limiting topical drift. |
| Alternative | 0.3 | 0.1 | Two skills solve the same subproblem via different implementations | Provides minimal backward mass, mainly to keep interchangeable options reachable. |

**Table 7.** GoS hyperparameters used in all reported experiments. Values are the defaults shipped in the released code and are held fixed across benchmarks, model families, and library sizes unless explicitly noted.

| Hyperparameter | Value | Role |
|---|---|---|
| PPR restart $\alpha$ | 0.2 | Teleport probability in Eq. (4); controls how much mass stays near the seeds. |
| PPR max iterations | 50 | Power-iteration cap for the diffusion in Eq. (4). |
| PPR tolerance | $10^{-6}$ | Convergence threshold on $\|\mathbf{s}^{(\ell+1)} - \mathbf{s}^{(\ell)}\|$. |
| Dependency I/O threshold $\zeta$ | 0.6 | Minimum schema-overlap score required to add a deterministic dependency edge. |
| Validation candidates per node $k$ | 8 | Top-$k$ candidates per node sent to the LLM relation validator. |
| Semantic seed pool | 20 | Dense-retrieval candidate count per query before merging. |
| Lexical seed pool | 20 | Lexical-retrieval candidate count per query before merging. |
| Seed set size | 5 | Number of merged seeds carried into graph diffusion. |
| Retrieval top-$N$ | 8 | Maximum number of hydrated skills returned to the agent. |
| Per-skill char budget | 2,400 | Truncation limit applied to each hydrated skill payload. |
| Global context char budget | 12,000 | Total character budget over the hydrated bundle. |

**Pipeline Summary.** GoS proceeds in two phases. Offline, it parses local skill packages into normalized nodes, adds dependency edges by I/O matching, and augments the graph with sparse workflow, semantic, and alternative relations. Online, it forms a hybrid semantic–lexical seed set, applies reverse-aware graph diffusion, and returns a reranked, budgeted bundle of execution-ready skills.

***Implementation Substrate.*** GoS is implemented on top of a graph-backed retrieval substrate for workspace management, vector indexing, and graph storage, while replacing document-centric assumptions with skill-specific parsing, relation induction, and agent-oriented hydration. Concretely, the system maintains an HNSW vector index over skill representations together with a typed directed graph whose vertices are normalized skills and whose edges carry relation labels, directional semantics, and scalar weights. This yields a retrieval substrate in which semantic proximity and structural connectivity can be combined inside a single inference-time pipeline rather than treated as disjoint retrieval regimes.

***Parser-First Skill Normalization.*** Each local skill package is first parsed deterministically from its primary `SKILL.md` file and nearby package structure. The parser extracts the canonical name and description from YAML frontmatter, collects explicit input and output fields, recovers domain tags, tooling, example tasks, compatibility notes, and allowed tools from both frontmatter and markdown sections, and resolves script entrypoints by scanning the local `scripts/` directory when present. It also materializes a stable local source path and a rendered snippet used later for retrieval and hydration. This parser-first design keeps node construction anchored in executable package structure rather than relying entirely on free-form semantic extraction.

***LLM-Assisted Semantic Completion.*** When package documentation is incomplete, GoS optionally performs a lightweight LLM pass over the full markdown body to recover retrieval-critical semantic fields, including capability summaries, inputs, outputs, domain tags, tooling, and example tasks. Importantly, this stage is constrained to normalize a single skill node and is not used to emit graph relations directly. In other words, the LLM here serves as a high-precision semantic completion module for node attributes rather than as an unconstrained graph-construction oracle. The inferred fields are then merged back with the deterministic parse, with the implementation favoring completed semantic fields only when they improve the retrieval representation.

***Typed Node Representation.*** After normalization, each skill is serialized into a node record that stores both structured lists and compact textual views. Besides canonical descriptive fields, the node retains raw skill content, rendered snippets, script entrypoints, and a stable `Source:` path. This dual representation is operationally important: the graph and vector index operate over normalized fields, whereas the final agent-facing bundle requires concise but directly usable payloads that can be opened inside the execution environment without path reconstruction.

***Directed Typed Relation Induction.*** The GoS graph is a typed directed graph rather than an undirected similarity graph. Dependency edges are induced deterministically by matching producer outputs against consumer inputs in both directions for

each candidate pair. An edge $u \rightarrow v$ therefore has explicit executable semantics: $u$ can plausibly provide an artifact consumed by $v$. Because I/O compatibility is asymmetric in general, this dependency structure cannot be reduced to undirected similarity without losing the notion of prerequisite direction.

Non-dependency relations, namely *workflow*, *semantic*, and *alternative*, are added through sparse LLM validation rather than dense all-pairs inference. For each node, GoS first forms a bounded candidate pool by combining lexical overlap, semantic neighbors from the vector index, and I/O-based candidate expansion. The LLM is then asked only to validate high-confidence relations inside this restricted pool. This two-stage design keeps graph construction tractable and biases the resulting graph toward precision rather than density.

***Hybrid Seeding at Query Time.*** At retrieval time, GoS does not rely on vector search alone. The system first optionally rewrites the raw task request into a compact query schema containing the goal, operations, artifacts, constraints, and high-value keywords. Semantic seeding is then obtained from nearest-neighbor search in embedding space, while lexical seeding is computed from token overlap over normalized node fields such as name, capability, I/O descriptors, tooling, example tasks, entrypoints, and snippets. These candidate pools are merged and reranked before graph diffusion, so the graph is seeded by a hybrid entry set rather than by a single retriever. In practice, this detail matters because the quality of the initial seeds strongly influences whether later graph expansion can recover the correct prerequisite chain.

***Reverse-Aware Structural Diffusion.*** Retrieval over the graph uses a Personalized PageRank-style diffusion operator constructed from the directed typed edges. The implementation first inserts forward transition mass along each stored edge, and then injects type-specific reverse transitions so that relevance can flow back from a matched high-level skill toward likely prerequisites. The reverse coefficients are largest for dependency edges and smaller for workflow, semantic, and alternative links, reflecting the fact that reverse traversal is most justified when recovering executable prerequisites. Operationally, this means the graph remains directed, but retrieval is explicitly reverse-aware. GoS therefore does not collapse the graph into an undirected graph; instead, it performs controlled backward propagation during scoring.

***Reranking and Budgeted Hydration.*** The stationary graph score is not exposed to the agent directly. After diffusion, GoS reranks candidate skills by combining graph relevance with field-level query evidence, then hydrates only the top skills into an agent-facing bundle under both per-skill and global context budgets. Each hydrated payload includes a concise skill rendering, relevant execution notes, and the original local source path. The retrieval output therefore functions as a bounded execution context rather than a generic search-result list. This final budgeted hydration step is essential for preserving the efficiency advantage over flat all-skills loading while still presenting enough structure for downstream execution.

***Section Summary.*** From an implementation perspective, GoS is best understood as a hybrid graph-construction and retrieval system: deterministic parsing and I/O matching provide a reliable executable backbone; optional LLM semantic completion improves node quality when documentation is incomplete; sparse LLM relation validation adds higher-order inter-skill structure; and reverse-aware graph diffusion converts a small hybrid seed set into a compact, more execution-complete bundle. These implementation choices are what instantiate the central claim of the paper that structural retrieval should recover not only relevant skills, but also the prerequisite context needed to use them effectively.

## C Prompt and Interface Examples

***Layered Prompt Design.*** GoS uses two prompt layers with deliberately separated responsibilities. The first layer operates *inside* the indexing and retrieval stack, where LLMs are used only for constrained normalization, optional query rewriting, and sparse relation validation. The second layer operates at the *agent interface*, where the environment prompt tells the downstream agent when to call retrieval, how to interpret the returned bundle, and how strongly to prefer reuse over open-ended exploration. This separation is methodologically important. Graph-side prompts determine what semantic structure enters the retrieval substrate, whereas agent-side prompts determine whether that retrieved structure is converted into a verifier-aligned execution plan.

***Presentation Goal.*** This section is not intended to enumerate full prompt templates. Instead, it exposes the narrow prompt fragments and interface rules that are most important for understanding the method. From a reviewer perspective, the key point is that GoS does not rely on unconstrained prompt engineering. The internal prompts are used to normalize or validate bounded objects, and the external interface is used to constrain downstream behavior once retrieval has occurred. Together, these prompts form an interface contract between offline graph construction and online execution.

***Internal Prompt A: Skill Semantic Completion.*** The semantic-completion prompt is intentionally narrow. It asks the model to normalize exactly one skill document and extract only retrieval-critical fields. In the implementation, the prompt

**Table 8.** Representative prompt and interface components in GoS. The table highlights the small set of prompt contracts that shape graph construction and downstream agent behavior.

| Component | Role | Key constraint | Intended effect |
|---|---|---|---|
| **Internal Prompt A** | | | |
| Skill semantic completion | Normalize one skill document into retrieval-critical fields. | Preserve canonical `name`/`description`; fill only node-local semantic fields; return an empty `edges` list. | Improve node quality when `SKILL.md` is incomplete while preventing relation hallucination. |
| **Internal Prompt B** | | | |
| Relation validation | Verify whether a bounded candidate pair should receive a typed edge. | Restrict outputs to {dependency, workflow, semantic, alternative}; preserve exact source/target names; emit nothing when uncertain. | Keep the graph sparse and precise instead of generating dense all-pairs links. |
| **Internal Prompt C** | | | |
| Query rewrite | Rewrite a raw request into a compact retrieval schema. | Extract `goal`, `operations`, `artifacts`, `constraints`, and `keywords` without redefining the task. | Improve seed-stage lexical and semantic coverage while preserving task intent. |
| **Agent Interface** | | | |
| Retrieval usage contract | Tell the downstream agent when retrieval must be called and how the returned bundle should be used. | Run `graphskills-query` first; read `Retrieval Status`; use exact `Source:` paths; prefer adapting retrieved scripts; prioritize verifier-minimal behavior. | Make retrieval operational immediately, so the bundle narrows search instead of serving as optional background context. |

explicitly preserves the canonical `name` and `description` when present, emphasizes high precision, and requires the returned `edges` list to be empty. This design reflects a conservative use of LLMs: the model is allowed to fill semantic gaps in node attributes, but not to invent graph structure. Operationally, this improves the quality of node representations used for indexing while avoiding a common failure mode in LLM-built graphs, namely relation over-generation. The prompt is therefore best understood as a constrained semantic completion module, not as a latent graph generator.

***Internal Prompt B: Relation Validation.*** The relation-validation prompt is invoked only after GoS has formed a small candidate pool using lexical overlap, semantic neighbors, and I/O-based expansion. The prompt defines four edge types: *dependency*, *workflow*, *semantic*, and *alternative*. It also explicitly instructs the model to prefer sparse, high-precision edges, to emit nothing when uncertain, and to preserve exact skill names in the `source` and `target` fields. This makes the prompt function more like a relation verifier than a free-form graph generator. In practice, this design is important because it limits graph density and preserves the typed semantics later used during reverse-aware diffusion.

**Prompt excerpt: skill semantic completion**

```
1. Extract exactly one skill node from the document.
3. Infer only retrieval-critical fields: capability, inputs,
   outputs, domain_tags, tooling, example_tasks.
6. Use high precision. If uncertain, leave a field empty.
7. Do not invent relationships. Return an empty `edges` list.
```

This excerpt illustrates the central design principle of the internal extraction prompt: GoS uses the LLM as a constrained semantic normalizer, not as an unconstrained graph author. For the appendix, the important point is not merely that an LLM appears in the pipeline, but that the allowable output space is sharply restricted to node-local semantic completion.

***Internal Prompt C: Query Rewrite.*** The optional query-rewrite prompt maps a raw task request to a compact retrieval schema with fields such as `goal`, `operations`, `artifacts`, `constraints`, and `keywords`. The prompt explicitly instructs the model not to invent benchmark-specific labels and to leave unclear fields empty. This is consistent with the retrieval objective in GoS: rewriting is used only to expose retrieval-critical technical terms such as file formats, APIs, protocols, and concrete operations. It is not intended to change the task itself. When rewriting is disabled or unavailable, the system falls back to deterministic lexical normalization, so query rewriting is a retrieval enhancement rather than a mandatory dependency. In other words, the prompt improves lexical and semantic coverage at the seed stage, but it is not allowed to redefine the problem.

***Agent Interface Prompt.*** In the SkillsBench GoS environment, the agent is instructed to begin with a targeted retrieval query built from the task goal, artifact or format, operation or API, and verifier-critical constraints. The interface then forces the agent to read the retrieval status before continuing. A `NO_SKILL_HIT` response means the agent must explicitly acknowledge that no relevant skill was found and proceed without claiming skill use. A `SKILL_HIT` response means the returned bundle should be treated as a narrowing device: the agent is told to use the returned local source paths, inspect scripts before implementing from scratch, and prioritize the shortest path to verifier pass. This interface design matters because the main quality difference is often not whether some relevant skill exists somewhere in the library, but whether the agent receives a compact, execution-ready bundle early enough to affect the trajectory. In that sense, the interface prompt is part of the method rather than a presentation detail.

**Prompt excerpt: agent-facing retrieval interface**

```
Before writing any code, run:
graphskills-query "goal + artifact/format + operation/API +
verifier-critical constraint"

- If Retrieval Status: NO_SKILL_HIT, proceed without claiming skill use.
- If Retrieval Status: SKILL_HIT, use retrieved skills only as constraints.
- Use the exact Source path already returned.
- Prefer adapting retrieved scripts over broader re-implementation.
```

This second excerpt shows that the agent-facing interface is itself part of the method. It does not merely expose a search command. It constrains when retrieval is called, how the returned bundle is interpreted, and how aggressively the downstream agent is allowed to branch away from authoritative local implementations. The resulting effect is to make retrieval operational rather than decorative: the bundle is meant to narrow the search space immediately, not merely provide optional background context.

***Section Summary.*** Taken together, these examples show that prompt design in GoS is not generic scaffolding. The internal prompts constrain how semantic structure enters the graph; the external interface constrains how retrieved structure enters the agent's working context. The two layers therefore form an interface contract between offline graph construction and online agent execution. This contract is especially important in our setting because many failures are not pure retrieval misses, but retrieval-hit trajectories in which the agent still drifts unless the interface strongly biases it toward verifier-minimal reuse. The appendix evidence should therefore be read as support for a broader methodological claim: in graph-augmented agent systems, retrieval quality depends not only on what is indexed, but also on how the retrieved structure is exposed and behaviorally enforced downstream.

## D Core Retrieval Pseudocode

***Presentation Goal.*** For completeness, we provide pseudocode for the two main algorithmic stages of GoS: offline graph construction and online structural retrieval. The presentation is intentionally close to the implementation, but abstracted enough to foreground the method rather than the surrounding engineering details.

***Implementation Correspondence.*** Algorithm 1 corresponds to the parser-first normalization and relation-induction logic in the GoS implementation. Algorithm 2 corresponds to the retrieval path and the reverse-aware Personalized PageRank utilities. In practice, these stages additionally include engineering details such as workspace bootstrapping, embedding-dimension checks, source-path rewriting for containerized environments, and context clipping under hard character budgets. We omit those details from the pseudocode because they are not conceptually central, but they are nevertheless important for stable end-to-end deployment.

**Input:** Local skill documents $C$, optional LLM services, linking budget $k$
**Output:** Typed directed graph $G = (V, E)$ and vector index over normalized skills
Initialize empty node set $V$ and edge set $E$.
For each skill document $d \in C$:
    Parse YAML frontmatter and markdown structure from `SKILL.md`.
    Extract deterministic fields: name, description, inputs, outputs, tags, tooling, `Source:` path, and script entrypoints.
    If retrieval-critical semantic fields are incomplete, run constrained semantic completion to fill capability, inputs, outputs, and example tasks.
    Serialize the result as a normalized skill node $v$ and add $v$ to $V$.
For each ordered pair of nodes $(u, v)$ in a bounded candidate pool:
    Compute producer–consumer overlap between outputs of $u$ and inputs of $v$.
    If overlap exceeds threshold, add typed dependency edge $u \rightarrow v$.
For each node $u$:
    Form a sparse candidate set using lexical similarity, semantic neighbors, and I/O-based expansion.
    Run constrained relation validation on this candidate set.
    Add validated workflow, semantic, and alternative edges to $E$.
Build an embedding index over the normalized node representations.
Persist the typed graph, vector index, and retrieval metadata to the GoS workspace.

**Algorithm 1.** Offline graph construction for GoS

**Input:** Query $q$, prebuilt GoS workspace, retrieval budget $\tau$
**Output:** Bounded execution bundle $B(q)$
Optionally rewrite $q$ into a compact schema containing goal, operations, artifacts, constraints, and keywords.
Retrieve semantic seed candidates from the vector index.
Retrieve lexical seed candidates from normalized node fields.
Merge the candidate pools and construct a seed distribution $\mathbf{p}$.
Build a typed transition matrix over the graph.
    Add forward transition mass for each stored edge.
    Add relation-specific reverse mass, with the largest reverse coefficient on dependency edges.
Run reverse-aware Personalized PageRank until convergence to obtain graph scores $\mathbf{s}^\star$.
Rerank candidate skills using graph score plus direct field-level evidence from the query.
Hydrate the ranked skills into agent-facing payloads with exact `Source:` paths and concise execution notes.
Truncate the hydrated bundle under per-skill and global context budgets.
Return retrieval status, ranked bundle summary, bounded agent-facing context, and graph evidence among selected skills when it fits the budget.

**Algorithm 2.** Online graph-based skill retrieval

## E Error Analysis

***Error Taxonomy.*** We distinguish retrieval-side errors from downstream execution failures, since these correspond to different limits of the overall system. A retrieval method can fail because it never surfaces the correct skill, because it retrieves an incomplete bundle that omits critical prerequisites, or because it produces a broadly adequate bundle that is subsequently misused by the downstream agent. Treating these failure modes separately is important for attributing gains correctly: GoS is designed to improve retrieval completeness, but it cannot by itself eliminate planning or execution failures once a bundle has already been provided. Across our trajectories, several of the most informative failures are not simple retrieval misses, but long-horizon search failures in which the correct general tool family is present and the agent still does not converge to a verifier-passing implementation.

**Table 9.** Primary error modes in GoS-style retrieval experiments. The table separates retrieval failures, downstream execution failures, and infrastructure issues.

| Error type | Typical symptom | Whether GoS helps |
|---|---|---|
| Retrieval miss | The correct skill exists in the library but is never surfaced, so the agent falls back to a from-scratch path. | Yes; better seed quality and graph completion can reduce this failure. |
| Partial retrieval | A topically relevant skill is retrieved, but a parser, setup routine, converter, or constraint-carrying prerequisite is absent. | Yes; this is the main failure mode GoS is designed to reduce. |
| Good retrieval, bad execution | The retrieved bundle is broadly adequate, but the agent still drifts, over-engineers, or mismatches the verifier. | Only indirectly; this is primarily a backbone or planning issue. |
| Infrastructure failure | Build, environment, or startup failures prevent a meaningful episode from occurring. | No; these are excluded from model-quality comparisons. |

***Retrieval Misses.*** A retrieval miss occurs when the correct skill exists in the repository but is not surfaced at all. In this regime, the agent is forced onto a generic from-scratch path, so any downstream failure should be attributed primarily to the retriever rather than to execution drift. Misses typically arise when the query language does not overlap strongly with the skill description, when the task is phrased around a downstream objective but the critical skill is an upstream parser or setup utility, or when semantic retrieval overweights topical similarity relative to executable relevance. A representative example is `dapt-intrusion-detection`. In the failed GoS-style trajectory, the agent issued `graphskills-query` but did not recover `pcap-analysis`, instead receiving an irrelevant bundle that included items such as `dc-power-flow` and `dialogue-graph`. By contrast, the stronger baseline trajectory opened `pcap-analysis` and reused its tested helper code. The resulting difference in behavior is characteristic of a true retrieval miss: once the relevant analysis skill is absent, the task degrades into from-scratch implementation and fails the verifier.

***Partial Retrieval.*** Partial retrieval is more subtle and often more important. Here, the retrieved bundle contains an obviously relevant high-level skill, but omits one or more prerequisite helpers needed for successful completion. In our setting, these missing items are often parsers, format converters, preprocessing utilities, setup routines, or constraint-carrying reference skills. This is precisely the regime in which graph-aware retrieval is intended to help: once a high-level skill is matched as a seed, reverse-aware propagation can recover supporting skills that are weak semantic matches to the raw query but strong structural neighbors in the skill graph. `earthquake-phase-association` illustrates the boundary of this idea. In the stronger all-skills trajectory, the agent assembled a coherent seismic stack including `gamma-phase-associator`, `obspy-data-api`, `seisbench-model-api`, and `seismic-picker-selection`, and the task passed. In the corresponding GoS case, the graph retrieval did bring in a partially relevant seismic bundle, but the resulting context was still less complete, and the task failed with reward 0.0. This suggests that structural retrieval helps only when the recovered neighborhood is sufficiently complete to support the downstream pipeline, not merely when one or two domain-relevant skills are present.

***Good Retrieval, Bad Execution.*** Some failures occur even when the retrieved bundle is broadly adequate. In these cases, the agent may still over-generalize the task, ignore a retrieved authoritative interface, implement unnecessary functionality, or fail to align with the verifier. These episodes matter because they bound what can be credited to retrieval alone. They also motivate the conservative agent-facing instructions used in our environments, which emphasize verifier-minimal solutions, direct use of returned local source paths, and avoidance of unnecessary branching. `energy-market-pricing` is a representative example: both all-skills and GoS had access to the relevant economic-dispatch / power-flow skill family and both eventually passed, but the all-skills trajectory required substantially more agent time before converging. This is not a retrieval miss; it is a difference in how efficiently a broadly adequate bundle is converted into a verifier-passing plan. Conversely, `adaptive-cruise-control` shows the opposite failure mode: the retrieved bundle included clearly relevant control skills such as `pid-controller`, `mpc-horizon-tuning`, `vehicle-dynamics`, and `simulation-metrics`, yet the run still finished with reward 0.0. In that case the failure is better described as long-horizon execution drift or verifier mismatch rather than poor retrieval.

A related example is `dialogue-parser`. Multiple conditions had access to relevant task structure, yet only the strongest GoS trajectory converted that context into a full pass, while other conditions remained at partial reward. This again indicates that the dominant bottleneck was not the absence of any relevant skill at all, but how effectively the agent translated the available skill context into the exact output expected by the verifier.

***Infrastructure Failures.*** Finally, a subset of observed failures are not retrieval failures at all, but infrastructure failures such as environment build issues, setup crashes, or startup timeouts before a substantive episode begins. These cases are methodologically important but conceptually distinct: they should be tracked for experiment hygiene and rerun logic, yet they should not be interpreted as evidence against the quality of the retrieval method or the underlying model. Representative examples include Docker / BuildKit failures such as `layer does not exist` on `dapt-intrusion-detection`, missing compiler toolchains for `obspy`-dependent tasks, and logging failures that leave some trials with incomplete session traces or null token fields. These episodes matter operationally because they require reruns and infrastructure fixes, but they are not evidence about the retrieval quality of GoS, vector retrieval, or all-skills. For this reason, we treat them as experiment-hygiene issues and exclude them from method-quality interpretation whenever possible.

## F Qualitative Analysis

***Section Framing.*** We next examine a set of trajectory-grounded qualitative cases and compare the concrete skill evidence that actually entered the agent's working context in each condition. Table 10 reports the skills that materially shaped each run: for GoS and Vector Skills, these are the skills surfaced by the retrieval call and then used downstream, while for *Vanilla Skills* they are the skills the agent explicitly opened from the mounted library. This keeps the comparison grounded in executed trajectories rather than hypothetical retrieval quality.

Across the cases below, we focus on a single question: does the method expose a compact, execution-ready bundle early enough to change the trajectory? The main qualitative difference is often not whether a relevant skill exists somewhere in the repository, but whether the agent receives the right subset in a form that can be operationalized under the task budget.

***Case Study 1: Pedestrian Traffic Counting.*** The clearest intermediate case in our trajectories is `pedestrian-traffic-counting`. The task requires frame extraction, reliable pedestrian counting, and structured output generation. GoS surfaced a compact visual pipeline centered on `gemini-count-in-video`, `video-frame-extraction`, and `openai-vision`, and achieved the strongest outcome among the three conditions (0.417). The *Vanilla Skills* baseline did eventually open relevant helpers, including `gemini-count-in-video`, `video-frame-extraction`, and `object_counter`, but reached only a partial score (0.267). The *Vector Skills* run performed worst (0.041): although it issued the retrieval call, the retrieved context was not converted into a workable plan. This example is useful because it is not a pure pass/fail contrast. *Vanilla Skills* does locate relevant functionality, but GoS exposes a smaller and more coherent bundle that appears easier to operationalize within the available task budget.

***Case Study 2: Flood Risk Analysis.*** The `flood-risk-analysis` task illustrates a different regime: both GoS and *Vanilla Skills* succeed, but GoS exposes the required chain with much less search friction. In this task the correct workflow is not generic time-series analysis; it is specifically the combination of `usgs-data-download` for measurements, `nws-flood-thresholds` for stage cutoffs, and `flood-detection` for aggregation and comparison. GoS surfaced exactly this bundle and passed with reward 1.0. The *Vanilla Skills* baseline also passed with reward 1.0, but only after the agent explicitly searched through the large mounted library and opened the same family of skills. *Vector Skills*, by contrast, issued the retrieval command but never translated retrieval into a usable flood-analysis bundle, and the run failed with reward 0.0. This case is useful because it does not primarily show a reward gap between GoS and *Vanilla Skills*; instead, it shows that when the right execution chain exists in the repository, GoS mainly helps by making that chain explicit earlier in the trajectory.

***Case Study 3: Travel Planning.*** The `travel-planning` task is informative precisely because all three conditions surfaced clearly relevant travel skills. In the GoS run, the retrieved context centered on `search-cities`, `search-accommodations`, `search-attractions`, `search-driving-distance`, and `search-restaurants`, which is very close to the intended tool chain for the task. The *Vanilla Skills* baseline likewise opened essentially the same family of skills after searching through the library. *Vector Skills* also surfaced and used this same cluster of `search-*` skills, and that run passed the verifier with reward 1.0. This example sharpens the qualitative claim of the paper. The advantage of GoS is not that flat semantic retrieval can never recover the correct skill family; rather, it is that GoS more reliably exposes a compact and coherent bundle early in the episode. When *Vector Skills* does succeed, as it does here, its behavior becomes qualitatively much closer to GoS than to a clean retrieval miss.

***Case Study 4: Network Intrusion Detection.*** A clean GoS-positive example is `dapt-intrusion-detection`. In this case, GoS surfaced `pcap-analysis` together with adjacent analysis helpers such as `pcap-triage-tshark` and `threat-detection`, and the task passed. By contrast, the corresponding vector condition retrieved unrelated automation-oriented skills rather

**Table 10.** Trajectory-grounded skill evidence from executed qualitative cases. USEFUL denotes skills that were clearly operationalized downstream; NOISY denotes retrieved or opened items that were tangential or not visibly used.

| Task | GoS Bundle | Vanilla Bundle | Vector Bundle |
|---|---|---|---|
| pedestrian-traffic-counting | USEFUL gemini-count-in-video; multimodal-fusion; openai-vision; video-frame-extraction<br>NOISY threat-detection | USEFUL gemini-count-in-video; object_counter; openai-vision; video-frame-extraction<br>NOISY alfworld-heat-object-with-appliance; alfworld-object-locator; broader noisy context | USEFUL none<br>NOISY google-classroom-automation; rdkit; salesforce-service-cloud-automation; segmetrics-automation |
| flood-risk-analysis | USEFUL flood-detection; nws-flood-thresholds; usgs-data-download<br>NOISY time_series_anomaly_detection; -21risk-automation | USEFUL flood-detection; nws-flood-thresholds; usgs-data-download | USEFUL none<br>NOISY leverly-automation; scienceworld-room-navigator; text-to-speech; broader noisy context |
| travel-planning | USEFUL search-accommodations; search-attractions; search-cities; search-driving-distance; search-restaurants<br>NOISY search-flights; fjsp-baseline-repair-with-downtime-and-policy | USEFUL search-accommodations; search-attractions; search-cities; search-restaurants | USEFUL search-accommodations; search-attractions; search-cities; search-driving-distance; search-restaurants<br>NOISY additional noisy items |
| dapt-intrusion-detection | USEFUL pcap-analysis; pcap-triage-tshark<br>NOISY dc-power-flow; power-flow-data; -21risk-automation | USEFUL no clearly reused core skill | USEFUL none<br>NOISY codacy-automation; jakarta-namespace; rootly-automation; broader noisy context |
| dialogue-parser | USEFUL dialogue_graph; webshop-query-parser<br>NOISY browser-testing; obj-exporter; alfworld-goal-interpreter | USEFUL dialogue_graph | USEFUL none<br>NOISY docnify-automation; scienceworld-task-focuser; temporal-python-testing; broader noisy context |
| earthquake-phase-association | USEFUL gamma-phase-associator; seisbench-model-api; seismic-picker-selection<br>NOISY flood-detection; -21risk-automation | USEFUL gamma-phase-associator; obspy-data-api; obspy-datacenter-client; seisbench-model-api; seismic-picker-selection<br>NOISY gamma-automation; seismic-automation | USEFUL none<br>NOISY fixer-automation; maven-build-lifecycle; segmetrics-automation; broader noisy context |
| energy-market-pricing | USEFUL dc-power-flow; power-flow-data; locational-marginal-prices; casadi-ipopt-nlp<br>NOISY -21risk-automation | USEFUL dc-power-flow; economic-dispatch | USEFUL power-flow-data<br>NOISY aryn-automation; moxie-automation; mural-automation; broader noisy context |
| 3d-scan-calc | USEFUL mesh-analysis; dyn-object-masks<br>NOISY scienceworld-circuit-builder; scienceworld-circuit-connector; scienceworld-conductivity-tester | USEFUL mesh-analysis<br>NOISY broader noisy context | USEFUL mesh-analysis; obj-exporter; pymatgen; threejs<br>NOISY reflow-profile-compliance-toolkit |
| adaptive-cruise-control | USEFUL imc-tuning-rules; pid-controller; safety-interlocks; vehicle-dynamics<br>NOISY -21risk-automation | USEFUL no clearly reused core skill | USEFUL pid-controller; mpc-horizon-tuning; integral-action-design; simulation-metrics; vehicle-dynamics |
| econ-detrending-correlation | USEFUL timeseries-detrending<br>NOISY artifacts-builder; dyn-object-masks; mesh-analysis; -21risk-automation | USEFUL no clearly reused core skill | USEFUL none<br>NOISY breezy-hr-automation; scienceworld-object-classifier; webshop-purchase-initiator; broader noisy context |

than a usable PCAP analysis bundle, while the all-skills condition still failed despite full library access. This case is a useful counterpart to the retrieval-miss pattern discussed in the Error Analysis section: once retrieval surfaces the right analysis bundle, the task becomes a reuse problem rather than a from-scratch reverse-engineering problem.

***Case Study 5: Dialogue Parsing.*** The `dialogue-parser` examples show a strong gradient across methods. GoS converted the task into a full pass while exposing a compact bundle centered on `dialogue_graph`, together with structural helpers such as `obj-exporter`, `browser-testing`, and parser-oriented support. *Vanilla Skills* eventually improved once it explicitly opened `dialogue_graph`, and *Vector Skills* also reached a substantial partial score, but neither condition showed the same level of structured completeness as the strongest GoS trajectory. This case illustrates a pattern that appears repeatedly: once the right latent-representation skill is surfaced early, the rest of the pipeline becomes much easier for the agent to operationalize.

***Case Study 6: Earthquake Phase Association.*** `earthquake-phase-association` is a useful negative case for GoS because it shows that structural retrieval does not help automatically when the recovered neighborhood is still incomplete. In the strongest all-skills trajectory, the agent assembled a seismic processing stack including `gamma-phase-associator`, `obspy-data-api`, `obspy-datacenter-client`, `seisbench-model-api`, and `seismic-picker-selection`, and the task passed. The corresponding GoS case surfaced only a weaker subset, centered on `gamma-phase-associator`, `seisbench-model-api`, and `seismic-picker-selection`, with an irrelevant distraction skill mixed into the bundle, and the task failed. This is exactly the kind of case that is easy to miss if one looks only at whether some domain-relevant skill was retrieved. The qualitative difference is that the all-skills trajectory assembled a more execution-complete stack, while the GoS trajectory remained one step short of the required pipeline.

***Case Study 7: Energy Market Pricing.*** A final useful case is `energy-market-pricing`, where all-skills and GoS both passed but with very different trajectory quality. The all-skills condition explicitly used `dc-power-flow` and `economic-dispatch`, while GoS surfaced a broader but still coherent optimization bundle including `dc-power-flow`, `power-flow-data`, `locational-marginal-pr` and `casadi-ipopt-nlp`. Both runs eventually passed the verifier, but the trajectory quality differed sharply: GoS converted the retrieved bundle into a solution with substantially less agent-side search. This is one of the clearest examples in which the main value of GoS is not higher reward, but a shorter path from retrieval to execution.

***Case Study 8: 3D Scan Calculation.*** The `3d-scan-calc` task serves as a useful control because all three conditions can succeed when they recover the same latent geometry bottleneck. GoS exposed `mesh-analysis` together with adjacent geometric helpers, directly matching the preprocessing structure of the task. *Vanilla Skills* also reached a passing solution once the agent opened `mesh-analysis`, but the surrounding library context was notably noisier. *Vector Skills* likewise passed when it surfaced `mesh-analysis` together with geometry-oriented companions such as `obj-exporter`, `pymatgen`, and `threejs`. The qualitative lesson is therefore not that GoS is uniquely capable of solving the task; rather, when all methods recover a geometry-centered bundle, all can succeed, and the remaining difference is how directly that bundle is exposed.

***Case Study 9: Adaptive Cruise Control.*** `adaptive-cruise-control` is a useful failure case because all three conditions surfaced highly plausible control-related skills and still failed. GoS exposed `imc-tuning-rules`, `pid-controller`, `safety-interlocks`, and `vehicle-dynamics`, while *Vector Skills* surfaced an even more explicit control bundle including `pid-controller`, `mpc-horizon-tuning`, `integral-action-design`, `simulation-metrics`, and `vehicle-dynamics`. The *Vanilla Skills* condition also had access to a broad control-oriented context, yet none of the three settings converged to a passing solution. This case is important precisely because it is not a retrieval miss. It shows that once the task requires a long control-design and verifier-alignment chain, even a qualitatively good skill bundle may not be enough; the dominant bottleneck shifts from retrieval to execution discipline.

***Case Study 10: Economic Detrending and Correlation.*** The `econ-detrending-correlation` task offers a complementary success case. GoS surfaced `timeseries-detrending` and converted the task into a full pass, while the all-skills condition failed to assemble a comparably coherent detrending-centered bundle. *Vector Skills* also reached a full pass, but with a noisier retrieved context whose skills were only weakly connected to the intended econometric workflow. This case is useful in two ways. First, it shows another task where surfacing the right latent preprocessing step, here detrending rather than raw correlation, materially changes the result. Second, it reinforces the lesson from `travel-planning`: vector retrieval can still succeed, but its successful episodes do not always arise from a bundle that is as semantically crisp or structurally interpretable as the one surfaced by GoS.

***Takeaway.*** Across all ten qualitative cases in this appendix, the main pattern is not simply that GoS retrieves skills with better topical overlap. Rather, GoS more often exposes a bundle that is already close to the executable decomposition of the task. The `pedestrian-traffic-counting` example shows a genuine middle case in which *Vanilla Skills* finds relevant tools but still underperforms the tighter GoS bundle. The `flood-risk-analysis` example shows that when the correct chain is available to multiple methods, GoS mainly reduces search friction and makes the intended execution path explicit earlier. The `travel-planning`, `3d-scan-calc`, and `econ-detrending-correlation` examples show that *Vector Skills* can also succeed when it recovers the right family of skills, but these successes are most convincing when the retrieved bundle becomes qualitatively similar to what GoS surfaces directly. The `dialogue-parser` and `dapt-intrusion-detection` cases show how GoS can convert that structural advantage into clearer downstream wins. By contrast, `earthquake-phase-association` shows a real boundary condition in which GoS still falls short of an execution-complete bundle, while `energy-market-pricing` and `adaptive-cruise-control` show that even with broadly adequate retrieval, trajectory efficiency and verifier alignment remain separate bottlenecks. Taken together, these cases support the core claim of the paper: structural retrieval helps not only by improving relevance, but by presenting agents with a more execution-ready context.