Group of Skills: Group-Structured Skill Retrieval for Agent Skill Libraries

Kun Zeng[♠]*, **Yu Huo**[♠]*, **Siyu Zhang**[♡♠], **Haoyue Liu**[♠], **Yuecheng Zhuo**[♢], **Zi Ye**[♣], **Yuquan Lu**[♣], **Junhao Wen**[♣], **Xiaoying Tang**[♠]†

[♠]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen
[♣]Sun Yat-sen University [♡]UC San Diego [♢]Taiyuan University of Technology

yuhuo03/Group-of-Skills yuhuo03/group-of-skills-data

Abstract

Skill-augmented agents increasingly rely on large reusable skill libraries, but retrieving relevant skills is not the same as presenting usable context. Existing methods typically return atomic skills or dependency-aware bundles whose internal roles remain implicit, leaving the agent to infer the execution entry point, support skills, visible requirements, and failure-avoidance guidance. We introduce **G**roup **o**f **Skills** (GoSkills), an inference-time group-structured retrieval method that changes the agent-facing retrieval object from a flat skill list to a compact, role-labeled execution context. GoSkills builds anchor-centered skill groups from a typed skill graph, expands support groups through a group graph, bottlenecks the selected group plan into a bounded set of atomic skill payloads, and renders a fixed execution contract with Start, Support, Check, and Avoid fields, without changing the downstream agent, skill payloads, or execution environment. Across SkillsBench, ALFWorld, and Terminal-Bench 2 with six backbones, GoSkills attains the highest mean reward or accuracy among the evaluated methods in every benchmark–backbone setting, and matched controls show higher reward and fewer agent turns with group selection than with fixed-contract or post-hoc role-annotation alternatives.

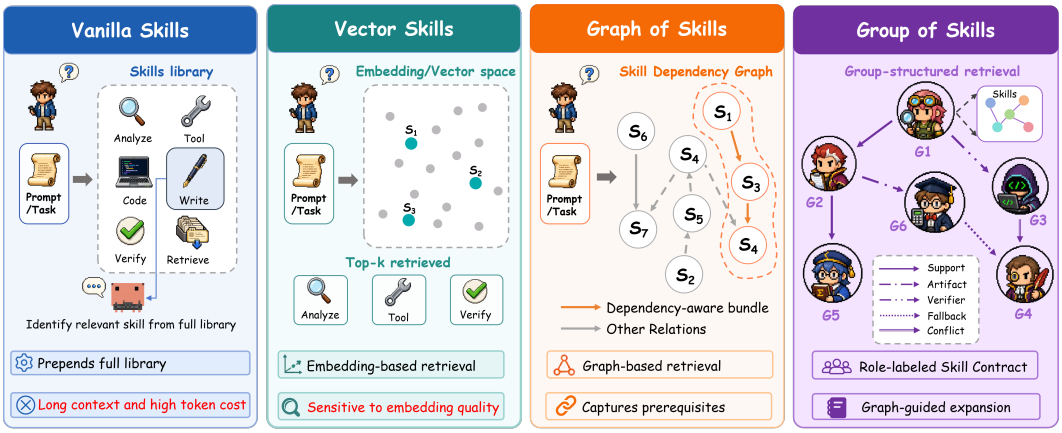

Figure 1: From individual skill retrieval to group-structured retrieval: Vanilla Skills expose the library directly, Vector Skills retrieve top-k similar skills, Graph of Skills hydrates dependency-aware bundles, and Group of Skills scores anchor-centered groups and expands them into a group plan.

*Equal contribution
†Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Skill-augmented LLM agents increasingly rely on external skill libraries: reusable snippets, procedural templates, tool instructions, checkers, and task-specific conventions that are too numerous to place in the prompt at once (Jiang et al., 2026; Wang et al., 2026; Han et al., 2025). This reliance on reusable external knowledge aligns with broader efforts to empower LLMs through knowledge integration at inference time (Wu et al., 2025). As these libraries grow, the bottleneck shifts from whether an agent can access skills to how retrieved skills should be organized under a limited context budget. Full-library prompting preserves recall but is expensive; flat semantic retrieval is cheaper but can miss functionally required skills; and graph-based retrieval improves recall by modeling relations among skills (Li et al., 2026a; Xia et al., 2026). Yet even a relevant retrieved bundle may still leave the downstream agent to infer the execution entry point, supporting skills, visible requirements, and failure-avoidance guidance. Figure 1 summarizes this progression from individual skills and hydrated bundles toward role-aware skill groups.

This motivates a different question: what unit should a skill retriever expose to the agent? Existing interfaces usually decide which atomic skills or hydrated bundles to include, while leaving the agent-facing roles among them implicit (Qu et al., 2024; Shi et al., 2025). In verifier-sensitive coding and interaction tasks, this missing organization can matter as much as recall: a setup utility may need to precede a checker, a parser may only be useful as support for a formatter, and visible requirements such as output formats or public tests must remain explicit after context compression (Hendrycks et al., 2021; Geng et al., 2023; Kang et al., 2025). We use "visible requirements" only for information available before execution, excluding hidden tests, evaluator internals, and previous failure traces.

We introduce *Group of Skills* (GoSkills), an inference-time group-level retrieval and contextualization method for agent skill libraries. The key idea is to change the agent-facing retrieval object from a flat list of atomic skills to a compact, role-labeled execution context. A useful group is not an arbitrary semantic cluster: it is an anchor-centered local pattern whose support members add complementary roles, artifact coverage, visible checks, or failure-avoidance cues. Offline, GoSkills constructs such bounded groups from typed skill neighborhoods and links related groups through a group graph. At inference time, it retrieves an anchor group, expands support groups, bottlenecks the selected group plan into a small set of atomic skill payloads, and renders a fixed execution contract. Because the downstream agent, skill payloads, and execution environment are unchanged, we isolate the intervention to retrieval-time context organization rather than agent training, tool execution, or environment modification.

We evaluate GoSkills on **SkillsBench** (Li et al., 2026c), which tests technical skill selection with visible, checkable requirements; **ALFWorld** (Shridhar et al., 2021), which tests multi-turn interactive decision-making; and **Terminal-Bench 2** (Merrill et al., 2026), which tests transfer to terminal tasks. Together, these benchmarks examine whether group-structured context helps agents use retrieved skills across technical problem solving and interactive execution under constrained skill budgets. A matched control study shows higher mean reward and fewer agent turns with group selection than with fixed-contract or post-hoc role-annotation controls, which is consistent with reduced organizational effort.

Our contributions are:

- We formulate *group-structured skill retrieval* for agent skill libraries: retrieval selects anchor-centered skill groups and expands them under a context budget, rather than exposing isolated skills or post-hoc bundles.
- We propose GoSkills, a deterministic inference-time method that decomposes skill-context construction into anchor selection, support expansion, and payload exposure. This decomposition makes retrieved skills directly usable by rendering the resulting group plan as an execution contract with coverage debt over visible requirements.
- We evaluate GoSkills on SkillsBench, ALFWorld, and Terminal-Bench 2. Results show higher mean reward or accuracy across all evaluated backbones, with benchmark-dependent token and runtime tradeoffs, supported by matched controls, task-level paired confidence intervals, and coverage diagnostics.

2 Related Work

Tool and skill retrieval. Tool-augmented language models, API-retrieval systems, and tool-use benchmarks show that external tools can expand agent capabilities, while large tool or skill collections make retrieval necessary (Schick et al., 2023; Mialon et al., 2023; Patil et al., 2024; Qin et al., 2024; Shi et al., 2025; Huo et al., 2026; Yao et al., 2023; Zhuang et al., 2023; Li et al., 2023). Machine memory research studies how external information can be encoded into computable representations that support learning and reasoning (Zheng et al., 2025). Prior skill repositories and benchmarks emphasize packaging, discovering, and evaluating reusable skills across diverse agent tasks and library settings (Wang et al., 2023; Zhang et al., 2026; Li et al., 2026c; Liang et al., 2026; Li et al., 2026b). However, retrieving relevant skills is not equivalent to presenting usable context: a retrieved bundle may still leave the execution entry point, support roles, visible constraints, and failure-avoidance guidance implicit. Our work targets this interface layer by using role structure during group selection and expansion, before deciding which payloads to expose to the agent.

Structured retrieval. Graph-structured retrieval has been studied for documents, memory, and tool access, where relations help retrieval move beyond independent nearest-neighbor matching (Edge et al., 2024; Gutierrez et al., 2024; Liu et al., 2024b,a). Hypergraph computation provides a related perspective on representing higher-order relations among multiple entities (Gao et al., 2024). In skill settings, such structure can recover prerequisites, setup utilities, preprocessors, and formatters that may not be lexically salient. GoSkills builds on this structural view but shifts the decision unit from individual skills to small role-aware groups, then expands an anchor group into support groups before bottlenecking the final payloads for the agent.

Concurrent work. Concurrent work uses structure at different stages of skill use. Graph of Skills retrieves dependency-aware execution bundles through graph construction, seeding, diffusion, reranking, and hydration (Li et al., 2026a), while GRASP studies structured skill composition and execution-time repair (Xia et al., 2026). In contrast, GoSkills uses structure before execution: it performs group-level retrieval and expansion, renders a role-labeled context, and leaves the downstream execution loop unchanged.

3 Methodology

Group of Skills (GoSkills) is an inference-time group-level retrieval and contextualization method for skill-augmented agents. Its goal is to change the agent-facing retrieval object from a flat list of atomic skills to a compact, role-labeled execution context. Unlike methods that stop after ranking or hydrating atomic skills, GoSkills uses atomic retrieval only as evidence for activating group-level retrieval units. As shown in Figure 2, GoSkills first builds reusable skill groups offline. At inference time, it retrieves an anchor group, expands it with support groups, bottlenecks the selected groups into a bounded set of atomic skill payloads, and renders them as an execution contract. Thus, any change in downstream behavior comes from how retrieved skills are organized and exposed, not from changing the agent, skill implementations, or environment.

We distinguish four objects. A *group* is an offline reusable local retrieval unit centered on a lead skill. A *group plan* $\mathcal{P}(q)$ is an ordered anchor-support structure selected for a query. $B(q)$ is the budgeted set of atomic skill payloads presented to the agent. $D(q, B)$ is any uncovered visible-requirement debt, and $C(q)$ is the deterministic contract rendered from $(\mathcal{P}, B, D)$. The contribution of GoSkills is the group-level retrieval, expansion, bottlenecking, and rendering policy.

3.1 Offline: Skill Groups and Group Graph

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a library of atomic skills. Each skill s has a hydrated payload and a normalized facet set F_s extracted from its name, metadata, tags, and text. We assume a typed skill graph

$$G_{\mathcal{S}} = (\mathcal{S}, E, w, \phi), \tag{1}$$

where edges encode dependency, workflow, semantic, artifact, or alternative relations.

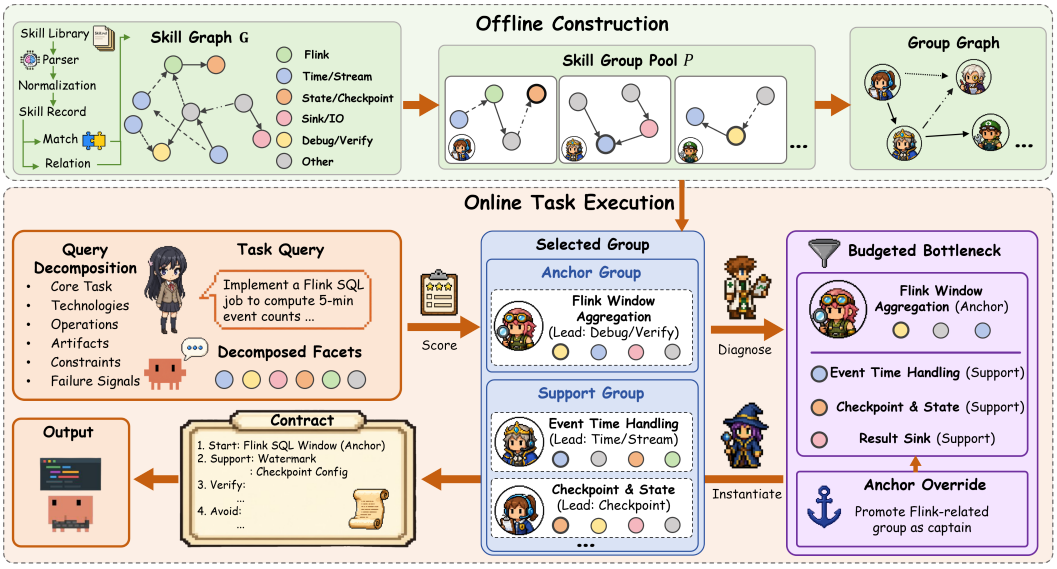

Figure 2: Overview of GoSkills. The offline stage constructs a skill graph from the skill library, extracts anchor-centered skill groups, and stores reusable group templates. At inference time, GoSkills decomposes the task query, retrieves and scores candidate groups, selects an anchor group, expands support groups, bottlenecks the selected group plan into a bounded set of atomic skill payloads, and renders a compact role-labeled execution contract for the downstream agent.

Offline, GoSkills constructs a pool $\mathcal{G}$ of small anchor-centered groups. Each group is represented as

$$g = \langle s_g^{\text{lead}}, M_g, R_g, F_g^+, F_g^{\text{opt}}, F_g^-, A_g, V_g, T_g, \pi_g \rangle, \tag{2}$$

where s_g^{lead} is the lead skill, M_g are at most two support members, R_g assigns member roles, F_g^+, F_g^{opt}, and F_g^- encode required, optional, and negative applicability facets, A_g and V_g store artifact and visible-requirement cues, T_g records the local topology, and π_g is a fixed group prior. A group is not an executable hidden program; it is a bounded role template whose members may later be expanded, pruned, and rendered under a prompt budget.

GoSkills also builds a typed group graph

$$\mathcal{H} = (\mathcal{G}, E_{\mathcal{G}}, \rho_{\mathcal{G}}, \omega), \tag{3}$$

where $\rho_{\mathcal{G}}$ labels each group edge as support, artifact, visible-check, fallback, or incompatibility evidence, and ω assigns edge weights. During online expansion, GoSkills follows only positive non-incompatibility edges:

$$\mathcal{N}_{\mathcal{H}}^+(\mathcal{P}, q) = \{g' \notin \mathcal{P}_{\text{set}} : \exists g \in \mathcal{P}_{\text{set}}, (g, g') \in E_{\mathcal{G}}, \rho_{\mathcal{G}}(g, g') \neq \textit{incompat}, \omega(g, g') > 0\}. \tag{4}$$

This graph lifts atomic skill relations to reusable group-level relations. Instead of repeatedly rediscovering support skills from the atomic graph at inference time, GoSkills can expand from an anchor group to nearby support groups that already encode role, artifact, and visible-requirement structure. In this way, the group graph turns low-level skill connectivity into a reusable retrieval substrate for agent-facing context organization. Construction details are given in Appendix B.2.

3.2 Online: Group Retrieval and Contextualized Exposure

Given a query q, GoSkills produces

$$\Omega(q) = \big(\mathcal{P}(q), B(q), D(q, B), C(q)\big). \tag{5}$$

The group plan is an ordered anchor-support object:

$$\begin{aligned} \mathcal{P}(q) &= \langle g_a(q), \mathcal{P}_{\text{sup}}(q) \rangle, \\ \mathcal{P}_{\text{set}}(q) &= \{g_a(q)\} \cup \mathcal{P}_{\text{sup}}(q). \end{aligned} \tag{6}$$

$B(q) \subseteq \mathcal{S}$ is the presented atomic skill set, $D(q, B)$ is remaining coverage debt, and $C(q)$ is the rendered execution contract. The plan is constrained by a context budget τ; in our implementation, payload count is the primary budget and character-level estimates act as a guard.

Query schema. The query is mapped to a deterministic schema

$$\psi(q) = \langle F_{\text{core}}, F_{\text{tech}}, F_{\text{op}}, F_{\text{artifact}}, F_{\text{constraint}}, F_{\text{failure}}, F_{\text{check}} \rangle . \tag{7}$$

Let

$$\mathcal{S}_g = \{s_g^{\text{lead}}\} \cup M_g. \tag{8}$$

The schema summarizes task goals, technology anchors, operations, artifacts, constraints, failure cues, and visible-requirement semantics. Visible requirements are task properties available before execution, such as exact output formats, required artifacts, deterministic behavior, unit tests, or formal proof obligations. In the released implementation, schema extraction uses the retrieval query (written by the agent or taken from the task instruction; Appendix E.3) and explicitly supplied public context. Skill metadata and text provide separate retrieval features. Hidden tests and evaluator internals are outside the query schema.

Group activation and scoring. Atomic retrieval evidence may be used, but it is not the final retrieved object. Let

$$R_0(q) = \{s_{(1)}, \ldots, s_{(M)}\} \tag{9}$$

be an optional seed set returned by a vector or graph retriever. Candidate groups are activated by direct query–group matches and by overlap with seed skills:

$$\mathcal{G}_q = \textsc{DirectGroupMatches}(\psi(q), \mathcal{G}) \cup \{g \in \mathcal{G} : \mathcal{S}_g \cap R_0(q) \neq \emptyset\}. \tag{10}$$

Thus $R_0(q)$ serves as evidence for group activation, while the final retrieval decision is made over groups rather than individual skills.

The selection policy is fixed and deterministic rather than learned. It uses three linear scores:

$$\begin{aligned} U_{\text{grp}}(g, q) &= \boldsymbol{\beta}^\top \mathbf{x}(g, q) + \lambda_\pi \pi_g, \\ U_{\text{sup}}(g \mid \mathcal{P}, q) &= \boldsymbol{\eta}^\top \mathbf{z}(g, \mathcal{P}, q), \\ U_{\text{bot}}(s \mid B, \mathcal{P}, q) &= \boldsymbol{\gamma}^\top \mathbf{h}(s, B, \mathcal{P}, q). \end{aligned} \tag{11}$$

where U_{grp} ranks candidate groups, U_{sup} adds marginal support groups, and U_{bot} selects final atomic skills for presentation. The feature families include retriever relevance, facet coverage, anchor match, visible-check support, graph connectivity, redundancy, negative applicability, and cost. These feature families define the scoring policy; Appendix D.1 specifies the coefficients and rules of the released implementation.

Anchor and support-group expansion. Candidate groups are ranked by U_{grp}, and the top $L(q)$ groups are retained in a shortlist $\widehat{\mathcal{G}}_q$, where $L(q)$ is a capped size determined by query complexity and retrieval ambiguity. Among admissible shortlisted groups (Appendix D.1), the anchor group is selected by

$$g_a = \arg\max_{g \in \widehat{\mathcal{G}}_q} \left[U_{\text{grp}}(g, q) + \lambda_a \text{Anchor}(g, q) \right], \tag{12}$$

where $\text{Anchor}(g, q)$ promotes lead skills matching explicit technology or artifact anchors and suppresses generic or incompatible leads. If no group is admissible, the plan is empty; otherwise, starting from $\mathcal{P}_0 = \langle g_a, \emptyset \rangle$, GoSkills greedily adds support groups from the retained shortlist and the group-graph neighborhood:

$$g^\star = \arg\max_{g \in \left(\widehat{\mathcal{G}}_q \cup \mathcal{N}^+_{\mathcal{H}}(\mathcal{P}, q)\right) \setminus \mathcal{P}_{\text{set}}} U_{\text{sup}}(g \mid \mathcal{P}, q). \tag{13}$$

Support expansion stops when the marginal gain, group budget, or context guard is exhausted. The implementation caps the selected group plan at three groups.

Bottlenecking and coverage-safe backfill. The selected group plan is internal. The downstream agent receives only $B(q)$ and the contract. Let

$$\mathcal{S}_{\mathcal{P}} = \bigcup_{g \in \mathcal{P}_{\text{set}}} \mathcal{S}_g. \tag{14}$$

GOSKILLS inserts lead skills first and then selects remaining presented skills with U_{bot}, subject to $\text{cost}(B) \leq \tau$. This bottleneck uses group structure for planning without exposing all group members.

To avoid losing explicit requirements, GOSKILLS computes coverage debt

$$D(q, B) = \mathcal{F}_{\text{high}}(q) \setminus \bigcup_{s \in B} F_s^{\text{vis}}(q), \tag{15}$$

where $\mathcal{F}_{\text{high}}(q)$ denotes high-confidence query facets such as framework names, file extensions, named APIs, output formats, and explicit constraints, and $F_s^{\text{vis}}(q)$ contains facets of the presented skill name and body, excluding hidden metadata and omitted text. Backfill uses only eligible activated skills that cover current debt, pass negative-applicability checks, and fit the remaining budget. Remaining debt is reported; accounting details are in Appendix B.4.

Execution contract. The final output is a deterministic execution contract

$$C(q) = \langle C_{\text{start}}, C_{\text{support}}, C_{\text{check}}, C_{\text{avoid}}, B(q), D(q, B) \rangle \,. \tag{16}$$

The contract names the anchor skill, labels support skills, lists visible requirements, states negative guidance, and includes the hydrated payloads for the presented atomic skills. The template is fixed across tasks; task-specific content enters only through the selected groups, presented skills, query schema, and coverage debt. GOSKILLS does not bind arguments, check preconditions, execute skills, or perform runtime graph repair.

Cost. Offline group construction runs once per skill library. Online, group activation uses inverted member and facet indexes under bounded selection and payload budgets; total cost also includes seed retrieval, which scans library documents per query for the default lexical seeder or inherits an external retriever's cost (complexity terms in Appendix B.6).

4 Experiments

4.1 Setup

We evaluate GOSKILLS on **SkillsBench**, **ALFWorld** and **Terminal-Bench 2**, covering both technical skill selection with deterministic checks and multi-turn interactive task execution. These benchmarks test whether group-structured retrieval improves agent-facing skill use under a constrained context budget. Within each benchmark–backbone comparison, methods share the agent loop, environment, and source skill payloads. The main comparison uses 2,000-skill libraries for all three benchmarks. Retrieved-skill methods share the exposure budget. Appendix E.16 specifies the settings.

Baselines. We compare against four skill-access settings: *No Skills*, which provides no skill context; *Vanilla Skills* (Anthropic, 2026), which prepends task-provided skills on SkillsBench and exposes the full library on ALFWorld and Terminal-Bench 2; *Vector Skills* (OpenAI, 2024), which retrieves a flat semantic top-k list; and *Graph of Skills* (Li et al., 2026a), which hydrates dependency-aware bundles from a typed skill graph. In contrast, GOSKILLS retrieves anchor-centered groups, expands support groups through a group graph, and renders the selected context as a role-labeled execution contract. Details are shown in Appendix E.13.

Models. We evaluate six backbones: GPT-5.4 (OpenAI, 2026), Claude Sonnet 4.5 (Anthropic, 2025), Gemini 3 Pro (Pichai et al., 2025), MiniMax M2.7 (MiniMax, 2026), Qwen 3.5 (Qwen Team, 2026), and Kimi K2.5 (Moonshot AI, 2026). Each model–method–benchmark combination is run three times, and Table 1 reports the mean over the three runs. Reward is averaged over tasks within each run and then averaged across runs. Reward is the primary outcome. Token usage reports mean cumulative input tokens over the agent loop, and runtime reports mean agent-only task-processing

Table 1: Aggregate downstream results by benchmark, method, and agent backbone. **Reward** (SkillsBench, ALFWorld) and **Accuracy** (Terminal-Bench 2) are averaged over tasks and reported in %, **Tokens** denotes mean cumulative input tokens, and **Runtime** denotes agent-only runtime (s). For **Reward/Accuracy**, larger is better; for **Tokens** and **Runtime**, smaller is better. Values are averaged over three repeats (sample SDs: Appendix E.17). Symbol columns indicate skill-library access (**Skill**), bounded retrieval (**Ret.**), and group-structured context (**Group**); ✓ indicates dependency-graph structure without group roles. Colored arrow deltas in skill-access rows are relative to No Skills under the same benchmark and backbone; green/red indicates improvement/regression under the metric direction. The best value for each metric within each benchmark–model block is highlighted in **bold** and the second best is underlined; No Skills is a no-context reference and participates in metric highlighting.

Model	Method	Access			SkillsBench			ALFWorld			Terminal-Bench 2		
		Skill	Ret.	Group	Reward ↑	Tokens ↓	Runtime ↓	Reward ↑	Tokens ↓	Runtime ↓	Accuracy ↑	Tokens ↓	Runtime ↓
GPT-5.4	No Skills	✗	✗	✗	25.3	**116,107**	188.6	83.6	**18,521**	35.3	52.8	**164,039**	**301.1**
	Vanilla Skills	✓	✗	✗	28.9 ↑ 3.6	121,275 ↑ 5.2k	**159.2** ↓ 29.4	92.5 ↑ 8.9	2,953,003 ↑ 2934.5k	74.4 ↑ 39.1	44.9 ↓ 7.9	2,071,755 ↑ 1907.7k	445.7 ↑ 144.6
	Vector Skills	✓	✓	✗	34.0 ↑ 8.7	144,827 ↑ 28.7k	185.8 ↓ 2.8	92.5 ↑ 8.9	38,263 ↑ 19.7k	38.5 ↑ 3.2	48.3 ↓ 4.5	177,525 ↑ 13.5k	357.7 ↑ 56.6
	Graph of Skills	✓	✓	✓	35.7 ↑ 10.4	136,054 ↑ 19.9k	192.9 ↑ 4.3	91.8 ↑ 8.2	39,013 ↑ 20.5k	40.2 ↑ 4.9	46.1 ↓ 6.7	168,981 ↑ 4.9k	302.7 ↑ 1.6
	GoSkills	✓	✓	✓	**39.4** ↑ 14.1	156,696 ↑ 40.6k	187.3 ↓ 1.3	**95.5** ↑ 11.9	35,042 ↑ 16.5k	**34.5** ↓ 0.8	**60.7** ↑ 7.9	173,228 ↑ 9.2k	427.6 ↑ 126.5
Gemini 3 Pro	No Skills	✗	✗	✗	20.1	**138,914**	161.7	82.1	**19,447**	**31.1**	49.4	**263,918**	**236.4**
	Vanilla Skills	✓	✗	✗	22.4 ↑ 2.3	148,217 ↑ 9.3k	**127.9** ↓ 33.8	88.3 ↑ 6.2	2,964,318 ↑ 2944.9k	58.7 ↑ 27.6	40.1 ↓ 9.3	1,934,487 ↑ 1670.6k	370.8 ↑ 134.4
	Vector Skills	✓	✓	✗	27.9 ↑ 7.8	165,931 ↑ 27.0k	168.4 ↑ 6.7	91.8 ↑ 9.7	43,016 ↑ 23.6k	36.6 ↑ 5.5	46.4 ↓ 3.0	300,841 ↑ 36.9k	257.3 ↑ 20.9
	Graph of Skills	✓	✓	✓	30.6 ↑ 10.5	167,643 ↑ 28.7k	174.1 ↑ 12.4	91.0 ↑ 8.9	37,491 ↑ 18.0k	33.2 ↑ 2.1	42.3 ↓ 7.1	279,723 ↑ 15.8k	265.1 ↑ 28.7
	GoSkills	✓	✓	✓	**33.0** ↑ 12.9	178,762 ↑ 39.8k	155.6 ↓ 6.1	**94.8** ↑ 12.7	35,829 ↑ 16.4k	31.4 ↑ 0.3	**54.3** ↑ 4.9	292,977 ↑ 29.1k	314.9 ↑ 78.5
MiniMax M2.7	No Skills	✗	✗	✗	12.8	404,872	418.3	41.5	**44,587**	74.3	42.3	741,520	453.6
	Vanilla Skills	✓	✗	✗	14.2 ↑ 1.4	**401,377** ↓ 3.5k	**381.3** ↓ 37.0	44.0 ↑ 2.5	4,297,163 ↑ 4252.6k	97.6 ↑ 23.3	35.2 ↓ 7.1	3,013,804 ↑ 2272.3k	608.3 ↑ 154.7
	Vector Skills	✓	✓	✗	15.9 ↑ 3.1	529,146 ↑ 124.3k	442.6 ↑ 24.3	50.5 ↑ 9.0	67,814 ↑ 23.2k	68.7 ↓ 5.6	36.3 ↓ 6.0	763,739 ↑ 22.2k	562.1 ↑ 108.5
	Graph of Skills	✓	✓	✓	18.7 ↑ 5.9	462,039 ↑ 57.2k	406.3 ↓ 12.0	54.5 ↑ 13.0	61,903 ↑ 17.3k	73.1 ↓ 1.2	38.6 ↓ 3.7	**711,886** ↓ 29.6k	**422.3** ↓ 31.3
	GoSkills	✓	✓	✓	**20.3** ↑ 7.5	573,284 ↑ 168.4k	438.7 ↑ 20.4	**56.2** ↑ 14.7	65,587 ↑ 21.0k	**62.9** ↓ 11.4	**45.3** ↑ 3.0	830,471 ↑ 89.0k	540.2 ↑ 86.6
Claude Sonnet 4.5	No Skills	✗	✗	✗	19.4	**398,761**	303.4	84.6	**15,846**	**34.8**	37.5	**571,286**	**392.6**
	Vanilla Skills	✓	✗	✗	23.6 ↑ 4.2	431,096 ↑ 32.3k	**243.1** ↓ 60.3	91.3 ↑ 6.7	3,206,972 ↑ 3191.1k	56.1 ↑ 21.3	31.1 ↓ 6.4	2,541,279 ↑ 1970.0k	635.6 ↑ 243.0
	Vector Skills	✓	✓	✗	26.4 ↑ 7.0	485,712 ↑ 87.0k	287.6 ↓ 15.8	92.8 ↑ 8.2	31,408 ↑ 15.6k	36.5 ↑ 1.7	32.2 ↓ 5.3	622,735 ↑ 51.4k	448.1 ↑ 55.5
	Graph of Skills	✓	✓	✓	27.1 ↑ 7.7	499,023 ↑ 100.3k	325.2 ↑ 21.8	96.3 ↑ 11.7	28,031 ↑ 12.2k	44.6 ↑ 9.8	33.0 ↓ 4.5	594,106 ↑ 22.8k	427.4 ↑ 34.8
	GoSkills	✓	✓	✓	**31.9** ↑ 12.5	521,658 ↑ 122.9k	284.9 ↓ 18.5	**97.0** ↑ 12.4	26,267 ↑ 10.4k	38.9 ↑ 4.1	**44.2** ↑ 6.7	674,148 ↑ 102.9k	573.7 ↑ 181.1
Qwen 3.5	No Skills	✗	✗	✗	17.2	**431,209**	494.1	65.9	**31,184**	89.2	35.6	**585,067**	756.4
	Vanilla Skills	✓	✗	✗	19.1 ↑ 1.9	440,257 ↑ 9.0k	**430.4** ↓ 63.7	76.4 ↑ 10.5	3,588,241 ↑ 3557.1k	181.4 ↑ 92.2	25.5 ↓ 10.1	2,836,437 ↑ 2251.4k	1067.1 ↑ 310.7
	Vector Skills	✓	✓	✗	25.3 ↑ 8.1	551,106 ↑ 119.9k	498.3 ↑ 4.2	74.4 ↑ 8.5	58,037 ↑ 26.9k	100.3 ↑ 11.1	32.6 ↓ 3.0	655,241 ↑ 70.2k	960.2 ↑ 203.8
	Graph of Skills	✓	✓	✓	25.0 ↑ 7.8	505,338 ↑ 74.1k	547.8 ↑ 53.7	78.4 ↑ 12.5	64,509 ↑ 33.3k	91.4 ↑ 2.2	28.5 ↓ 7.1	590,953 ↑ 5.9k	**734.2** ↓ 22.2
	GoSkills	✓	✓	✓	**29.6** ↑ 12.4	594,291 ↑ 163.1k	503.6 ↑ 9.5	**80.6** ↑ 14.7	53,912 ↑ 22.7k	**80.8** ↓ 8.4	**38.6** ↑ 3.0	625,988 ↑ 40.9k	938.4 ↑ 182.0
Kimi K2.5	No Skills	✗	✗	✗	18.6	**410,538**	361.2	78.1	**26,183**	**61.3**	36.3	**626,205**	**423.4**
	Vanilla Skills	✓	✗	✗	22.8 ↑ 4.2	434,619 ↑ 24.1k	**295.7** ↓ 65.5	84.8 ↑ 6.7	3,612,027 ↑ 3585.8k	149.7 ↑ 88.4	29.6 ↓ 6.7	2,668,963 ↑ 2042.8k	648.2 ↑ 224.8
	Vector Skills	✓	✓	✗	26.1 ↑ 7.5	508,245 ↑ 97.7k	336.3 ↓ 24.9	88.6 ↑ 10.5	55,471 ↑ 29.3k	63.4 ↑ 2.1	30.7 ↓ 5.6	657,488 ↑ 31.3k	469.6 ↑ 46.2
	Graph of Skills	✓	✓	✓	28.9 ↑ 10.3	448,172 ↑ 37.6k	358.9 ↓ 2.3	87.8 ↑ 9.7	50,836 ↑ 24.7k	73.8 ↑ 12.5	32.2 ↓ 4.1	676,329 ↑ 50.1k	436.6 ↑ 13.2
	GoSkills	✓	✓	✓	**31.2** ↑ 12.6	545,407 ↑ 134.9k	351.1 ↓ 10.1	**92.5** ↑ 14.4	53,459 ↑ 27.3k	65.3 ↑ 4.0	**42.7** ↑ 6.4	632,436 ↑ 6.2k	583.8 ↑ 160.4

time, excluding environment setup. Hosted-API runtime remains sensitive to provider load; agent-turn counts provide complementary evidence of organizational effort. Appendix E.1 describes the benchmark protocol, while Appendix E.8 reports valid-run denominators and aggregation provenance for the entries in Table 1.

4.2 Main Results

We present the main results in Table 1 and Figure 3; run provenance and valid-run denominators are reported in Appendix E.8. Across SkillsBench, ALFWorld, and Terminal-Bench 2, GoSkills improves over the *No Skills* setting for all evaluated backbones, showing that the retrieved group-structured context provides useful task guidance. It also exceeds *Vanilla Skills* on every backbone and benchmark. Relative to *Graph of Skills*, its agent-only runtime is lower in five of six SkillsBench and all six ALFWorld cells and higher in all six Terminal-Bench 2 cells.

SkillsBench. On SkillsBench, GoSkills obtains the highest mean reward for every backbone. Under GPT-5.4, reward increases from 35.7 to 39.4 and runtime drops from 192.9s to 187.3s. Under Claude Sonnet 4.5, reward increases from 27.1 to 31.9 and runtime drops from 325.2s to 284.9s.

These results suggest that the gain comes not only from retrieving relevant skills but also from presenting them usably. SkillsBench tasks often carry explicit artifacts, output formats, or public checks. A dependency-aware bundle may contain useful skills yet leave the entry point and support roles implicit; the START, SUPPORT, CHECK, and AVOID fields make them explicit, reducing the burden of reorganizing context.

ALFWorld. GoSkills obtains the highest reward across the evaluated backbones. Compared with Graph of Skills, reward increases by 0.7–4.7 percentage points and agent-only runtime decreases

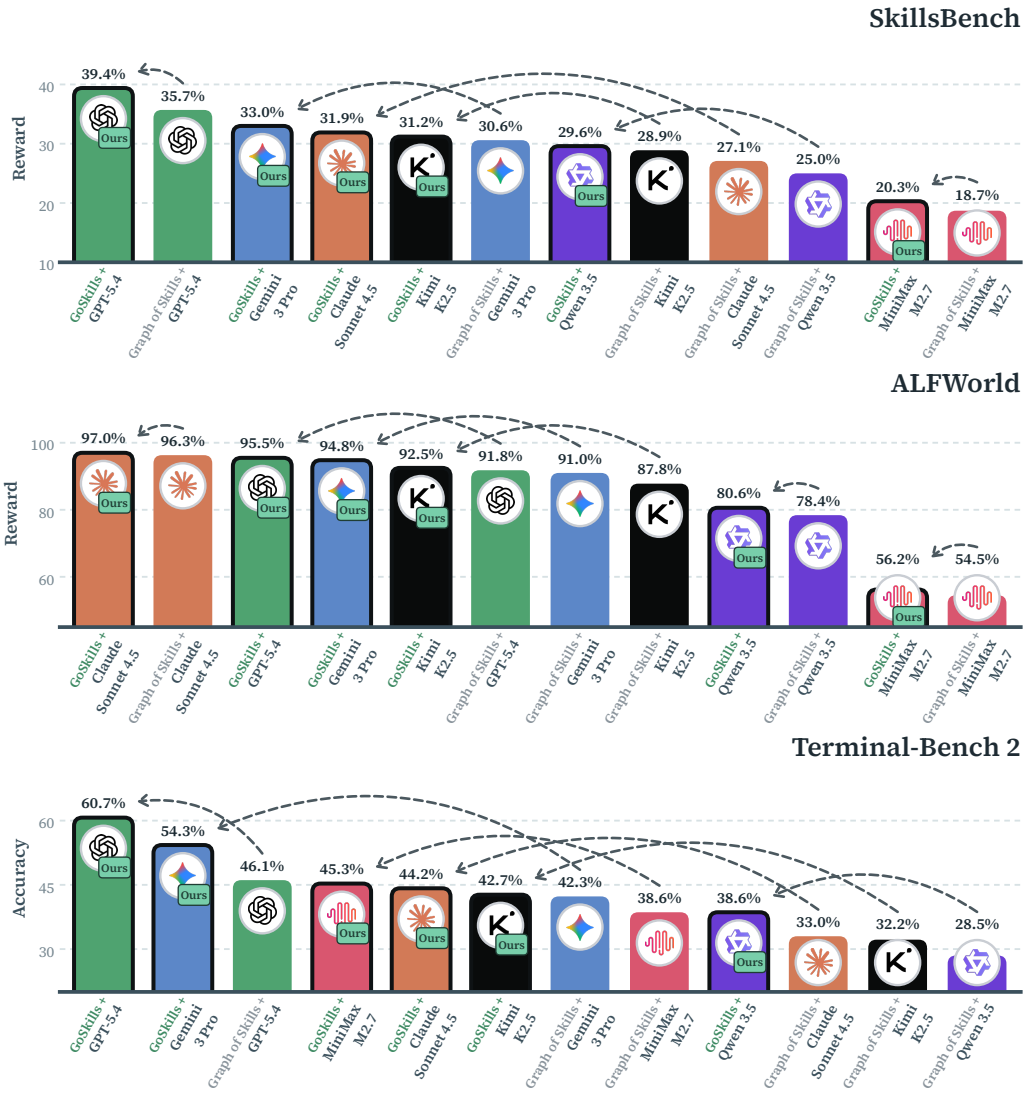

Figure 3: GOSKILLS versus Graph of Skills under each agent backbone. Rows report **SkillsBench** reward, **ALFWorld** reward, and **Terminal-Bench 2** accuracy; bars within a row are sorted by score. Bar color and logo identify the backbone, outlined bars marked "Ours" are GOSKILLS, and each dashed arrow links a backbone's Graph of Skills result to its GOSKILLS result. The reward and accuracy axes do not start at zero. Results for all methods are in Table 1.

across all six backbones. The smaller absolute margins reflect headroom: No Skills already reaches 65.9–84.6 on five of six backbones, where GOSKILLS removes 43–81% of the remaining error. ALFWorld also exposes fewer visible technical requirements and no deterministic checks, so the CHECK, AVOID, and debt fields carry less signal.

Terminal-Bench 2. Terminal-Bench 2 requires multi-step terminal execution, debugging, and verification. Across all six evaluated backbones, GOSKILLS achieves the highest accuracy and improves over *Graph of Skills* by 6.7–14.6 percentage points. Agent-only runtime increases across all six backbones, and cumulative input tokens increase for five of six (Table 1). Explicit entry points, support roles, and checks may help agents organize execution. However, the off-domain 2,000-skill library may lack task-specific capabilities, and accuracies under our fixed agent loop are not comparable to the public leaderboard (Appendix E.7).

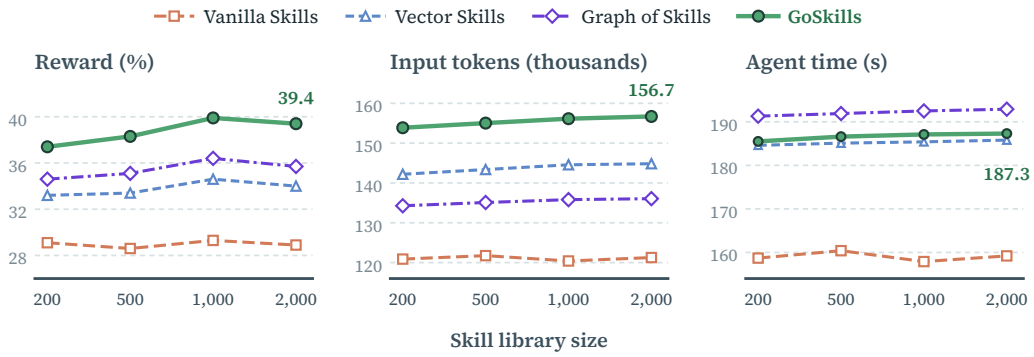

Figure 4: Sensitivity to library size on SkillsBench under GPT-5.4. Points show mean reward, cumulative input tokens, and agent-only time across three repeats at four library sizes. GoSkills has the highest mean reward at each size, while its observed runtime remains relatively stable.

Contribution of group selection. Table 2 presents our primary comparison for assessing group selection on all 87 SkillsBench tasks under GPT-5.4; all methods share the skill payloads, agent, environment, and retrieval budgets. Relative to Graph of Skills with the fixed contract, GoSkills improves reward by 2.2 percentage points (paired 95% CI $[-3.9, 8.8]$) and uses 0.9 fewer turns (95% CI $[-2.0, 0.2]$). The post-hoc (PH) controls assign roles to the top retrieved skills with a rule-based (PH-Rule) or LLM-based (PH-LLM) annotator and perform requirement backfill without constructing groups. Both have lower mean reward and more turns than full GoSkills, consistent with a benefit from group selection beyond contract formatting and post-hoc role annotation (Appendix E.5).

Retrieval-gate diagnostic. The must-hit column uses 40 annotated visible-requirement items from the 10-task `tasks-extra` development subset, separately from the 87-task end-to-end evaluation. GoSkills achieves a 1.00 must-hit rate with fewer than four presented skills on average, a development-set diagnostic of coverage under the bottleneck rather than held-out evidence (both gate modes in Appendix E.10).

5 Ablation Study

Sensitivity to library size. Figure 4 compares libraries of 200 to 2,000 skills on SkillsBench under GPT-5.4. GoSkills has the highest reward at each size, peaking at 39.9 with 1,000 skills and reaching 39.4 with 2,000. Its agent time ranges from 185.5 to 187.3 seconds. Vanilla Skills uses task-provided skills on SkillsBench, so its variation across sizes reflects run-to-run noise.

Component ablations. Table 2 reports component ablations and a contract-matched control. The control keeps the Graph of Skills payloads fixed and renders them with the GoSkills contract. Reward rises from 35.7 to 37.2, while runtime falls from 192.9s to 190.6s. Full GoSkills reaches 39.4 reward and 187.3s.

The control study also compares post-hoc role annotation and requirement backfill under matched budgets. Among individual component removals, group expansion and coverage backfill produce the largest reward drops. Removing coverage backfill lowers must-hit from 1.00 to 0.73, while removing role labels or the Avoid field preserves must-hit. Retrieved Skills Only has a reward of 36.9 and must-hit of 0.76.

Robustness to imperfect library structure. With corruption applied before indexing and group construction, and no retuning, GoSkills retains 91.9% of clean reward under 50% metadata blanking and still leads Graph + Fixed Contract by 0.6 points. With raw names and descriptions only, GoSkills retains 91.4% of clean reward, with a 0.7-point margin. Removing 50% of atomic typed edges retains 97.7%, reflecting the additional metadata evidence for group-level relations; edge

Table 2: GPT-5.4 SkillsBench controls and ablations on 87 tasks with three repeats. Reward is in percent, tokens are cumulative input tokens, and runtime is agent-only seconds. Must-hit is measured on 40 development-gate items. Arrows show differences from full GoSkills (green/red for improvement/regression, $k = 10^3$ tokens). Bold marks the best values, including ties. Sample SDs are in Appendix E.17.

Method	Reward ↑	Tokens ↓	Runtime ↓	Turns ↓	Must-hit ↑
Graph of Skills	35.7 ↓ 3.7	**136,054** ↓ 20.6k	192.9 ↑ 5.6	15.8 ↑ 1.3	0.84 ↓ 0.16
Graph of Skills + GoSkills Contract	37.2 ↓ 2.2	144,517 ↓ 12.2k	190.6 ↑ 3.3	15.4 ↑ 0.9	0.84 ↓ 0.16
PH-Rule + backfill	36.7 ↓ 2.7	146,208 ↓ 10.5k	193.4 ↑ 6.1	15.6 ↑ 1.1	0.81 ↓ 0.19
PH-LLM + backfill	37.4 ↓ 2.0	147,376 ↓ 9.3k	195.2 ↑ 7.9	15.5 ↑ 1.0	0.83 ↓ 0.17
GoSkills	**39.4**	156,696	**187.3**	**14.5**	**1.00**
w/o Anchor Selection	38.2 ↓ 1.2	155,283 ↓ 1.4k	189.1 ↑ 1.8	14.7 ↑ 0.2	0.95 ↓ 0.05
w/o Group Expansion	37.5 ↓ 1.9	150,139 ↓ 6.6k	190.4 ↑ 3.1	14.8 ↑ 0.3	0.90 ↓ 0.10
w/o Group Graph	38.3 ↓ 1.1	154,626 ↓ 2.1k	189.5 ↑ 2.2	14.6 ↑ 0.1	0.96 ↓ 0.04
w/o Role Labels	38.6 ↓ 0.8	153,558 ↓ 3.1k	190.9 ↑ 3.6	14.8 ↑ 0.3	**1.00** = 0.00
w/o Coverage Backfill	37.8 ↓ 1.6	151,492 ↓ 5.2k	191.3 ↑ 4.0	14.9 ↑ 0.4	0.73 ↓ 0.27
w/o Avoid Field	38.8 ↓ 0.6	154,071 ↓ 2.6k	188.8 ↑ 1.5	14.6 ↑ 0.1	**1.00** = 0.00
Retrieved Skills Only	36.9 ↓ 2.5	146,903 ↓ 9.8k	192.1 ↑ 4.8	15.3 ↑ 0.8	0.76 ↓ 0.24

removal does not remove the group graph itself. Documentation quality is therefore an important operating condition. Appendix E.6 reports the complete sweep and seed variability.

6 Limitations

GoSkills is limited to inference-time retrieval and contextualization over a skill library. It does not train the downstream model, execute tools, bind arguments, or adapt after failures; when a required capability is absent from the library or disconnected from the group graph, the method can only report remaining coverage debt. The implementation depends on deterministic schema extraction and on skill metadata and typed relations constructed from the library; the corruption study shows graceful degradation, but documentation quality remains an operating condition, and the fixed scoring coefficients were set on a small development subset. It may be less effective on tasks with long setup chains, sparse metadata, ambiguous requirements, or hidden constraints outside the prompt. Efficiency gains are benchmark-dependent: relative to Graph of Skills, agent time decreases in five of six SkillsBench and all six ALFWorld settings but increases on Terminal-Bench 2 with an off-domain library. Three repeats per cell also limit run-to-run uncertainty estimates.

7 Conclusion

This paper introduces group-structured skill retrieval, which changes the retrieval unit from atomic skills to role-aware skill groups rendered as a compact execution contract, while leaving the downstream model and execution loop unchanged. Across the three benchmarks, GoSkills achieves higher mean reward or accuracy than the evaluated baselines, and matched controls attribute part of the gain to group selection beyond contract rendering. Its runtime advantage is benchmark-dependent, but the GPT-5.4 SkillsBench controls show fewer agent turns; we therefore read the efficiency result as reduced organizational effort rather than lower token cost. Overall, the results suggest that skill retrieval should consider not only which skills are relevant, but also how those skills form a usable execution structure under a limited context budget. Future work could replace the current deterministic scoring rules with learned group-selection policies, improve robustness to sparse or noisy skill metadata, and extend group construction to settings with implicit requirements and runtime feedback.

Acknowledgments and Disclosure of Funding

This work is supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant No. 2025A1515012968, in part by the Shenzhen Science and Technology Program under Grant No. JCYJ20240813113502004, in part by the National Natural Science Foundation of China under Grant No. 62001412, in part by Shenzhen Stability Science Program 2023, in part by the Guangdong Provincial Key Laboratory of Future Networks of Intelligence (Grant No. 2022B1212010001), and in part by the Shenzhen Key Lab of Crowd Intelligence Empowered Low-Carbon Energy Network (Grant No. ZDSYS20220606100601002). The authors declare no competing interests.

References

Anthropic. Introducing Claude Sonnet 4.5. `https://www.anthropic.com/news/claude-sonnet-4-5`, September 2025. Accessed: 2026-05-05.

Anthropic. Agent Skills. `https://github.com/agentskills/agentskills`, 2026. GitHub repository. Accessed: 2026-05-06.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Yue Gao, Shuyi Ji, Xiangmin Han, and Qionghai Dai. Hypergraph computation. *Engineering*, 40: 188–201, 2024.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952, 2023.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=hkujvAPVsg`.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. LEGOMem: Modular procedural memory for multi-agent LLM systems for workflow automation. *arXiv preprint arXiv:2510.04851*, 2025.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c24cd76e1ce41366a4bbe8a49b02a028-Paper-round2.pdf`.

Yu Huo, Siyu Zhang, Kun Zeng, Yuquan Lu, Cheng Yang, Yifu Guo, and Xiaoying Tang. RepoShapley: Shapley-enhanced context filtering for repository-level code completion. *arXiv preprint arXiv:2601.03378*, 2026.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic skills–beyond tool use in LLM agents. *arXiv preprint arXiv:2602.20867*, 2026.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. *arXiv preprint arXiv:2510.00615*, 2025.

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

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

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, 2023.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026c.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. SkillNet: Create, evaluate, and connect AI skills. *arXiv preprint arXiv:2603.04448*, 2026.

Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. ToolNet: Connecting large language models with massive tools via tool graph. *arXiv preprint arXiv:2403.00839*, 2024a.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, et al. ControlLLM: Augment language models with tools by searching on graphs. In *European Conference on Computer Vision*, pages 89–105. Springer, 2024b.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

MiniMax. MiniMax M2.7: Early echoes of self-evolution. `https://www.minimax.io/news/minimax-m27-en`, March 2026. Accessed: 2026-05-05.

Moonshot AI. Kimi K2.5: AI that sees, codes, and works like an expert. `https://www.kimi.com/ai-models/kimi-k2-5`, January 2026. Accessed: 2026-05-05.

OpenAI. text-embedding-3-large Model. `https://developers.openai.com/api/docs/models/text-embedding-3-large`, 2024. Accessed: 2026-05-06.

OpenAI. GPT-5.4 Model. `https://developers.openai.com/api/docs/models/gpt-5.4`, 2026. Accessed: 2026-05-05.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/e4c61f578ff07830f5c37378dd3ecb0d-Paper-Conference.pdf`.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. A new era of intelligence with Gemini 3. `https://blog.google/products-and-platforms/products/gemini/gemini-3/`, November 2025. Accessed: 2026-05-05.

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. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=dHng2O0Jjr`.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1930–1940, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026. Accessed: 2026-05-05.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1258. URL https://aclanthology.org/2025.findings-acl.1258/.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Fei Wu, Tao Shen, Thomas Bäck, Jingyuan Chen, Gang Huang, Yaochu Jin, Kun Kuang, Mengze Li, Cewu Lu, Jiaxu Miao, et al. Knowledge-empowered, collaborative, and co-evolving AI models: The post-LLM roadmap. *Engineering*, 44:87–100, 2025.

Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. GraSP: Graph-structured skill compositions for LLM agents. *arXiv preprint arXiv:2604.17870*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026.

Qinghua Zheng, Huan Liu, Xiaoqing Zhang, Caixia Yan, Xiangyong Cao, Tieliang Gong, Yong-Jin Liu, Bin Shi, Zhen Peng, Xiaocen Fan, et al. Machine memory intelligence: Inspired by human memory mechanisms. *Engineering*, 2025.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. ToolQA: A dataset for LLM question answering with external tools. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=pV1xV2RK6I.

A Appendix Overview

The appendix complements the main paper along four axes: method specification, implementation reproducibility, retrieval analysis, and trajectory-grounded qualitative analysis. Table 3 summarizes the role of the main appendix sections.

Table 3: Appendix roadmap. Each section is written to support a specific claim or reproducibility requirement from the main paper.

Section	Purpose
Additional Method Details	Defines the retrieval unit, offline group construction, online expansion, bottlenecking, coverage-debt accounting, contract template, and complexity terms.
Prompt and Interface Examples	Shows the fixed interface contract exposed to the downstream agent and explains how schema, support roles, checks, avoid rules, and remaining debt enter the prompt.
Implementation and Hyperparameters	Reports the fixed budgets, thresholds, feature families, coefficients, and deterministic operator rules used by GoSkills.
Additional Experimental Details	Documents the benchmark protocol, SkillsBench retrieval-gate analysis, task-level retrieval examples, retrieval controls, corruption and transfer studies, paired trajectory results, and token/runtime accounting.
Failure and Error Analysis	Separates retrieval misses, partial coverage, downstream execution failures, context overhead, and infrastructure failures.
Qualitative Analysis	Provides trajectory-grounded qualitative examples comparing Graph of Skills baseline trajectories with GoSkills retrieval and paired trajectory evidence.
Assets and Licenses	Lists every benchmark, harness, baseline, skill library, and model API used, with source, version, and license.

B Additional Method Details

This section provides implementation-level details for GoSkills. The main text describes the offline/online structure and the key retrieval decisions. The executable details below specify the released implementation: deterministic group construction, selection, source-preserving presentation, and contract rendering. These retrieval operators do not depend on the downstream backbone.

B.1 Comparison of Retrieval Units

Table 4: Comparison of agent-facing retrieval units. GoSkills differs from post-hoc bundle annotation by using role-aware groups before expansion and bottlenecking.

Method	Retrieval object	Expansion object	Role timing	Agent-facing output
Graph of Skills	Skill nodes scored by graph diffusion	Hydrated dependency-aware bundle	Mostly implicit or rendered after retrieval	Bounded skill bundle
Bundle reranking	Candidate skill set	Set-level rerank or pruning	Attached after set selection	Role-labeled bundle
GoSkills	Small local asymmetric group	Group-to-group support expansion	Used before scoring, expansion, and bottlenecking	Role contract plus payloads

B.2 Offline Group Construction

The offline stage constructs a reusable pool of small skill groups before any test-time query is observed. As shown in Algorithm 1, its purpose is not to change the underlying skill graph, but to summarize local typed neighborhoods into compact, role-annotated group candidates that can later

be retrieved and bottlenecked online. For each skill, we treat it as a potential lead skill, enumerate bounded groups within its typed neighborhood, assign intra-group roles, and extract the facets, artifact signatures, and visible-check cues used by the online contextualizer. We reject explicit member incompatibilities, require novelty from the second support in a triple, and deduplicate by lead and members. The resulting pool $\mathcal{G}$, graph $\mathcal{H}$, and member/facet indices I are reused across queries. This group-layer constructor consumes the supplied atomic graph; it does not query an LLM or reconstruct that graph online.

Algorithm 1 BUILDSKILLGROUPPOOL

Require: skill graph $G_{\mathcal{S}} = (\mathcal{S}, E, w, \phi)$, group size cap $K_{\max}$, neighbor cap $n_{\max}$
Ensure: group pool $\mathcal{G}$, group graph $\mathcal{H}$, inverted index I

$\mathcal{G} \leftarrow \emptyset, \quad I \leftarrow \emptyset$
for each lead skill $s \in \mathcal{S}$ **do**
  $N_s \leftarrow \text{TYPEDNEIGHBORHOOD}(s, E, n_{\max})$
  $P_s \leftarrow \text{ENUMERATEGROUPS}(s, N_s, K_{\max})$
  for each candidate group $g \in P_s$ **do**
    $R_g \leftarrow \text{ASSIGNROLES}(g, E, \phi)$
    $F_g^+, F_g^{\text{opt}}, F_g^-, A_g, V_g \leftarrow \text{EXTRACTGROUPFACETS}(g)$
    if $\text{COMPATIBLE}(g)$ **and** $\text{NONREDUNDANT}(g, \mathcal{G})$ **then**
      $\mathcal{G} \leftarrow \text{CANONICALINSERT}(\mathcal{G}, g)$
    end if
  end for
end for
$I \leftarrow \text{UPDATEINDEX}(\mathcal{G})$
$\mathcal{H} \leftarrow \text{BUILDGROUPGRAPH}(\mathcal{G}, E, \phi)$
return $\mathcal{G}, \mathcal{H}, I$

B.3 Online Contextualization Algorithm

At inference time, GOSKILLS extracts a schema from explicitly supplied public task text and activates groups through seed-member and query-facet indices. It selects an anchor and bounded support groups, reserves eligible leads and query-relevant incoming prerequisites, and then fills the remaining atomic-skill budget. Backfill can cover residual high-confidence requirements using already activated skills. Finally, source-preserving presentation prepares the payloads, recomputes debt, and renders the fixed contract. No step executes a skill or changes the task environment. If no anchor qualifies, backfill can still select debt-covering activated skills while the group plan stays empty.

Here q includes the caller's query and any explicitly supplied public context. The core does not discover task files automatically. Skill metadata and full skill text supply offline retrieval features, not requirements in $\psi(q)$. Selection initially uses bounded source prefixes to check visible coverage and the full rendered-context budget. Final presentation may substitute source sections while preserving every high-confidence facet previously credited to a prefix. Initialization rejects an empty contract that already exceeds the budget. In Algorithm 2, $\mathcal{S}_{\mathcal{P}}$ denotes the union of current plan members; all admission operators check the full rendered context, not just the prospective skill body.

B.4 Coverage-Debt Accounting

Let B_{out} and D_{out} be the outputs of Algorithm 2. The coverage debt is

$$D_{\text{out}} = \mathcal{F}_{\text{high}}(q) \setminus \bigcup_{s \in B_{\text{out}}} F_s^{\text{vis}}(q).$$

Here $F_s^{\text{vis}}(q)$ is extracted from the skill name and actual presented body, excluding hidden metadata and omitted source text. During selection and backfill, the same rule is applied to bounded source prefixes. Final presentation preserves their credited high-confidence facets and may cover additional ones; debt is recomputed afterward. Backfill stops when no eligible fitting candidate covers current debt, the skill budget is full, or its cap is reached. The output records uncovered requirements without promising a second search after presentation. Coverage and debt measure lexical evidence in the delivered text, not semantic sufficiency, task success, or execution verification. Preserving credited

Algorithm 2 GOSKILLS group retrieval and expansion.

Require: public query/context q, pool $\mathcal{G}$, graph $\mathcal{H}$, indices I, seed mode R_0, budget τ, backfill cap $c_{\max}$
Ensure: group plan $\mathcal{P}$, presented skills B, remaining debt D, execution contract C

$\psi \leftarrow \textsc{ExtractSchema}(q), \quad \mathcal{F}_{\text{high}} \leftarrow \textsc{HighConfidenceFacets}(\psi)$
$R_q^0 \leftarrow \textsc{RetrieveSeedSkills}(R_0, q), \quad B \leftarrow \emptyset$
$\mathcal{G}_q \leftarrow \textsc{CandidateGroups}(I, R_q^0, \psi)$
$\mathcal{A}_q \leftarrow R_q^0 \cup \bigcup_{g \in \mathcal{G}_q} \mathcal{S}_g$
$\widehat{\mathcal{G}}_q \leftarrow \textsc{TopGroups}(\mathcal{G}_q, U_{\text{grp}}, \psi, R_q^0)$
$g_a \leftarrow \textsc{SelectAnchor}(\widehat{\mathcal{G}}_q, \psi, \tau)$; initialize $\mathcal{P}$ to $[g_a]$, or [] if no group is admissible
while $\mathcal{P} \neq []$ **and** $|\mathcal{P}| < P_{\max}$ **do**
  $\mathcal{E}_t \leftarrow (\widehat{\mathcal{G}}_q \cup \textsc{GroupNeighbors}(\mathcal{H}, \mathcal{P})) \setminus \mathcal{P}$
  $g^\star \leftarrow \textsc{BestSupport}(\mathcal{E}_t, \mathcal{P}, \psi, R_q^0, \tau)$
  if $g^\star = \perp$ **then**
    break
  end if
  Append $g^\star$ to $\mathcal{P}$; $\mathcal{A}_q \leftarrow \mathcal{A}_q \cup \mathcal{S}_{g^\star}$
end while
$B \leftarrow \textsc{InsertReserved}(\mathcal{P}, R_q^0, \tau)$
while $|B| < b$ **do**
  $s^\star \leftarrow \textsc{BestSkill}((\mathcal{S}_{\mathcal{P}} \cap \mathcal{A}_q) \setminus B, B, \mathcal{P}, \psi, \tau)$
  if $s^\star = \perp$ **then**
    break
  end if
  Append $s^\star$ to B
end while
$D \leftarrow \textsc{CoverageDebt}(\mathcal{F}_{\text{high}}, B), \quad c_{\text{back}} \leftarrow 0$
while $D \neq \emptyset$ **and** $|B| < b$ **and** $c_{\text{back}} < c_{\max}$ **do**
  $s^\star \leftarrow \textsc{BestBackfill}(\mathcal{A}_q \setminus B, D, B, \mathcal{P}, \psi, \tau)$
  if $s^\star = \perp$ **then**
    break
  end if
  Append $s^\star$ to B; $c_{\text{back}} \leftarrow c_{\text{back}} + 1$
  $D \leftarrow \textsc{CoverageDebt}(\mathcal{F}_{\text{high}}, B)$
end while
$\mathcal{P} \leftarrow \textsc{AnchorPrune}(\mathcal{P}, B, \psi)$
Prepare source-preserving payloads for B; recompute visible facets and D
$C \leftarrow \textsc{FormatContract}(\mathcal{P}, B, D, \psi)$; reject if the full rendering exceeds τ
return $\mathcal{P}, B, D, C$

facets prevents hidden or truncated text from receiving coverage credit; it does not certify that the instructions are correct.

B.5 Execution Contract Template

The released implementation's execution contract template is fixed across tasks. Task-specific content enters only through the selected groups, presented skills, query schema, and coverage debt.

Table 5: Fixed execution-contract fields rendered by GOSKILLS. The field names remain constant across tasks; only the selected content changes.

Field	Rendered role
`START`	Anchor skill with a fixed rationale, or the no-anchor fallback
`SUPPORT`	Support skills with roles such as prerequisite, parser, formatter, checker, or fallback
`CHECK`	Visible output formats, artifacts, tests, deterministic constraints, or proof obligations
`AVOID`	Explicit query and selected-skill negative-applicability facets
`SKILLS`	Hydrated payloads for the presented atomic skills
`DEBT`	Remaining high-confidence visible requirements, if any

B.6 Complexity Details

Let $M = |R_0(q)|$, $\bar{\iota}$ be the average member-index posting length, $J_q = \sum_{f \in \psi(q)} |I_f|$ the total query-facet posting length, L the shortlist size, $P_{\max}$ the group cap, $K_{\max}$ the member cap, and b the presented-skill budget. After seed retrieval, candidate activation costs $O(M\bar{\iota} + J_q)$, and ranking costs $O(|\mathcal{G}_q| \log |\mathcal{G}_q|)$. With bounded facet and payload operations, support expansion costs $O(P_{\max}(L + P_{\max}\bar{d}_{\mathcal{H}})K_{\max})$, where $\bar{d}_{\mathcal{H}}$ bounds the visited group-graph degree. Bottlenecking costs $O(bP_{\max}K_{\max})$; backfill additionally scans the activated skill set up to $c_{\max}$ times. Presentation examines selected source sections and their static dependency closures; final rendering is linear in the emitted context.

These are group-stage costs. Default lexical seeding also tokenizes library documents, computes document frequencies, scores them, and sorts matched documents on each query. For total processed library text/term volume T, vocabulary size V, and $N = |\mathcal{S}|$, a conservative seed-stage bound is $O(T \log V + N \log N)$. Thus complete online cost includes a full-library lexical stage. External seed mode inherits the supplied retriever's cost. Offline enumeration is bounded by the two-neighbor cap; graph construction uses indexed typed-edge lifts and an exact similarity join whose cost remains data-dependent.

C Prompt and Interface Examples

Interface design. GOSKILLS uses a fixed downstream interface rather than a task-specific prompt template. Released lexical seeding splits hyphen and underscore separators to expose retrieval aliases, without rewriting the schema or adding requirements. Agent-facing content is determined by the group plan, presented source text, coverage debt, and fixed fields. This section shows the small set of interface components that matter for reproducibility. The purpose is not to claim that prompting alone is the method; rather, the prompt is the rendering surface through which group-level retrieval is made operational. This interface is not tuned per task. We do not manually edit START, SUPPORT, CHECK, AVOID, or DEBT fields for individual examples; the fields are filled deterministically from the selected group plan, bottlenecked skill payloads, query schema, and coverage-debt accounting.

Fixed prompt skeleton. Figure 5 shows the schematic field layout. In the released renderer, the apparent reason slots use fixed wording, not generated explanations: START states "Use ⟨name⟩ first because its group matches the visible task evidence." and asks the agent to inspect the source path. Each SUPPORT item gives a name and role followed by "supports the selected plan or an explicit requirement." CHECK lists sorted query check, artifact, and constraint facets; AVOID lists sorted query and selected-skill negative facets. SKILLS repeats the name, source path, and source-preserving body of each presented skill; DEBT lists sorted uncovered high-confidence facets. Without an anchor, START says "No eligible anchor fits this query and budget. Solve using the visible task requirements." Empty SUPPORT, SKILLS, and DEBT use "None."; empty CHECK uses "Follow the explicit task requirements."; empty AVOID uses "None identified from explicit applicability cues."

Illustrative contract interpretation. Figure 6 paraphrases the roles of the fields for an invoice analysis example. This prose is illustrative, not a verbatim renderer output. The executable template in

Table 6: Interface components exposed by GOSKILLS. Task-specific content enters through retrieved groups, presented skills, and visible requirements; the field structure is fixed.

Component	Role	Constraint
`START`	Names the anchor skill with a fixed evidence-based rationale.	Exactly one primary entry point is rendered when a selected anchor contributes a presented skill.
`SUPPORT`	Lists support skills and their roles, such as parser, formatter, checker, prerequisite, or fallback.	Only skills selected by bottlenecking or backfill are exposed; unpresented group members are not silently implied.
`CHECK`	States visible output formats, artifacts, deterministic checks, or proof obligations.	Lists query-schema check, artifact, and constraint facets. This differs from the high-confidence subset used for debt and excludes hidden evaluator state.
`AVOID`	Surfaces explicit negative-applicability facets.	Lists query negatives and selected-skill negatives. The text is guidance; it does not execute a runtime check.
`DEBT`	Reports high-confidence visible requirements still uncovered by the final presented skills.	Remaining debt is exposed instead of being silently repaired beyond the backfill budget.

Figure 5 uses fixed rationale sentences and extracted facet identifiers; it does not generate task-specific explanations.

Relation to retrieval. This interface differs from post-hoc bundle annotation because role labels are used before the final payload is rendered. The selected groups influence anchor choice, support-group expansion, bottlenecking, backfill, and the contract fields. As a result, the downstream agent receives a small set of atomic skill payloads together with the retrieval decision's intended execution structure. This is the mechanism evaluated by the retrieval-gate and case-study analyses.

D Implementation and Hyperparameters

This appendix records the released implementation profile of the method in Section 3. It specifies the frozen code's choices for seeds, offline construction, admission rules, coverage accounting, and presentation. These retrieval operators do not depend on the downstream backbone.

Table 7: Core settings of the released implementation. The profile fixes these values across queries.

Component	Setting
Seed retrieval evidence	Binary TF–IDF cosine by default; scored evidence with provenance in external mode
Group retrieval object	Anchor-centered skill groups
Group expansion	Group graph over support, artifact, visible-check, and fallback relations
Seed mode	Lexical retrieval by default
Returned skill budget	`top-n`=4
Seed skill count	`seed-top-k`=4
Maximum skill payload	1,800 characters
Maximum rendered context	9,000 characters
Maximum group size	3 skills
Maximum selected groups	3 groups
Backfill cap	At most 2 activated skills covering current visible debt
Rendered output	Execution contract + presented skill payloads + remaining debt
Downstream agent	Unchanged

GoSkills Execution Contract

```
START
Use [anchor_skill] first because [anchor_reason].
Inspect its source path before writing new code.
SUPPORT
Use these skills only for the stated roles:
[support_skill_1]: [role_1] -- [reason_1]
[support_skill_2]: [role_2] -- [reason_2]
CHECK
Before finalizing, verify these visible requirements:
[visible_format_or_artifact]
[visible_test_or_constraint]
AVOID
Do not follow these incompatible interpretations:
[negative_cue_1]
[negative_cue_2]
SKILLS
[hydrated_payloads_for_presented_skills]
DEBT
[remaining_coverage_debt_or_None]
```

Figure 5: Schematic execution-contract layout. The implementation substitutes names, roles, facet lists, and source payloads mechanically; reason placeholders use the fixed wording specified in the text.

D.1 Scoring Weights and Rule Definitions

All scores used by GoSkills are deterministic weighted sums over normalized features. Each scalar feature is clipped to $[0, 1]$ before scoring. The feature vectors $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{h}$ follow the column order in Table 10; stage-inapplicable features are set to zero. Penalty terms use negative coefficients, and rows are not constrained to sum to one because feature families are computed on different clipped scales. The coefficients in Table 10 comprise the 24 fixed coefficients of this profile. Lexical dictionaries, selection predicates, and presentation policy are versioned with the source code.

Feature normalization. Seed relevance is normalized by the largest returned seed score. Group relevance is the maximum over its members; support relevance uses only members outside the current plan; bottleneck relevance is the individual skill score. Coverage and check features measure the fraction of the corresponding query-facet set matched by novel offline facets; an empty requested set scores zero. They guide ranking but do not certify rendered coverage. Lead/skill anchor match is 1 for an explicit high-confidence technology or artifact match, 0.7 for a normalized anchor match, and 0 otherwise. Connectivity sums applicable positive internal edge weights; support selection takes the maximum of that value and the strongest outgoing group-edge weight from the plan. Redundancy divides already covered candidate facets by all candidate facets. Negative applicability is binary and also acts as a hard eligibility filter. Cost divides the rendered candidate payload length, including its skill name and source path, by 9,000. All eight values are clipped to $[0, 1]$. The group score additionally uses 0.05 times its clipped prior (the mean positive internal edge weight); anchor correction adds 0.18 times anchor match.

Adaptive shortlist. Let c be the fraction of the seven query-schema categories that are nonempty. From normalized seed scores, let Δ and s be the top-two gap and full spread; both are 1 when fewer than two seeds are returned. Define

$$a = 0.55(1 - \mathrm{clip}(\Delta)) + 0.45(1 - \mathrm{clip}(s)), \qquad d = 0.60c + 0.40a.$$

Illustrative instance: `invoice-fraud-detection` **(paraphrased)**

START
Use `fuzzy-match` as the anchor skill because the task requires detecting suspicious invoice entries under approximate entity and string matching.

SUPPORT
`pdf-reading`: extract invoice text and table fields.
`xlsx`: parse structured transaction records and preserve row-level identifiers.

CHECK
Preserve invoice IDs; produce the required structured output; verify suspicious entries against the visible task constraints.

AVOID
Do not treat exact string mismatch alone as fraud evidence. Do not ignore missing or malformed invoice fields.

SKILLS
Hydrated payloads for `fuzzy-match`, `pdf-reading`, and `xlsx`.

DEBT
Any explicit requirement absent from the presented source text remains listed.

Figure 6: Illustrative paraphrase of contract semantics for an invoice task, in the layout of Figure 5. These explanations are not literal renderer output.

Table 8: Group-pool and selection constants of the released profile. Lexical dictionaries are versioned with the implementation.

Parameter	Value
Complexity weight	0.60
Ambiguity weight	0.40
Ambiguity gap / spread weights	0.55 / 0.45
Base pool cap minimum	6
Top-n pool multiplier	2
Adaptive extra base	1.0
Adaptive difficulty multiplier	2.0
Candidate pool cap	32
Score-floor center	0.55
Score-floor difficulty slope	0.30
Score-floor absolute minimum	0.10
Minimum required floor / ceiling	3 / 6
Group selection minimum score	0.14
Support group / bottleneck skill minimum	0.10 / 0.10
Inter-group affinity threshold	0.35
Offline neighbor cap	2
Prior coefficient λ_π	0.05
Anchor correction λ_a	0.18

The cap is $L = \min\{32, \lceil \max(6, 2b) + 1 + 2d \rceil\}$, with score floor $\max\{0.10, 0.55 - 0.30d\}\max_{g \in \mathcal{G}_q} U_{\text{grp}}(g, q)$ (zero for no candidates). Among the first L ranked groups, retain those above the floor and always retain the first $\min\{L, \lceil 3 + 3d \rceil\}$, subject to availability. Ties use lead seed rank, smaller group size, and stable group ID in that order.

Operator semantics. Tables 11 and 12 define the deterministic operators of Algorithms 1 and 2 for the released profile. For an atomic-edge lift, affinity is the maximum of its clipped weight and the full positive-facet Jaccard similarity between groups. Semantic and alternative relations are symmetric; other positive atomic relations preserve direction. Dependency, workflow, and semantic edges lift to support, and alternative edges to fallback. Metadata-only links use a rare-first inverted-prefix

Table 9: Deterministic feature families used by the group, support, and bottleneck scores.

Feature family	Definition
Retriever relevance	Normalized seed-retrieval score of matched members
Facet coverage	Novel query-facet overlap with offline group/member facets for scoring; visible text alone determines coverage debt
Anchor match	Exact or normalized match to technology and artifact anchors
Visible-check support	Coverage of tests, formats, artifacts, or proof obligations
Connectivity	Internal typed-edge support, or its maximum with a group-edge weight from the plan during support selection
Redundancy penalty	Candidate-facet overlap with selected members' offline facets
Negative applicability	Explicit negative/high-confidence or anchor/query-negative conflict
Cost penalty	Candidate rendered-payload characters divided by the full-context cap

Table 10: Fixed coefficients for the three scoring stages. Rel. denotes seed or group relevance, Facet denotes query-facet coverage, Anch. denotes anchor match, Check denotes visible-check support, Conn. denotes graph connectivity, Redun. denotes redundancy, Neg. denotes negative applicability, and Cost denotes payload cost.

Score	Rel.	Facet	Anch.	Check	Conn.	Redun.	Neg.	Cost
U_{grp}	0.28	0.22	0.18	0.12	0.10	-0.05	-0.25	-0.04
U_{sup}	0.12	0.28	0.06	0.16	0.16	-0.18	-0.25	-0.04
U_{bot}	0.18	0.24	0.12	0.20	0.08	-0.12	-0.30	-0.08

candidate join followed by exact Jaccard verification at 0.35. A qualifying pair receives symmetric support links, plus artifact/check links when those sets overlap. Sharing a format alone does not bypass the full-facet threshold.

Grounded support and reservation. A candidate member has support evidence if it has positive seed relevance, adds a visible technology/operation/failure/check facet beyond current plan members, or is a positive dependency/workflow predecessor of a candidate-group member having either form of query evidence. A newly introduced mandatory lead must itself satisfy this predicate; if the lead is already a plan lead, evidence from an added member suffices. Shared artifacts or generic constraints alone are insufficient. Separately, the candidate must add a query-matching offline facet or have a positive group connection from the plan and at least one new member. These predicates precede scoring. Reserved predecessors are narrower: they connect into a selected lead, carry prerequisite/preprocessor roles, and have positive seed relevance. Candidate support groups must also be pairwise compatible with all members of the existing plan.

Seeds and source-preserving presentation. Default seeds use binary TF–IDF cosine over public skill documents, with $\mathrm{idf}(t) = 1 + \log((1+N)/(1+\mathrm{df}(t)))$ for N documents. Both query and document lengths are normalized; ties use skill ID. Hyphens and underscores are split only for retrieval aliases; no model rewrites the query. External mode requires explicit nonnegative scored skill IDs with source provenance in a separate configuration.

After selection, complete source sections are ranked by query overlap, relevant workflow or imperative cues, and deterministic tie-breakers. The extractor includes ancestor headings and statically identified earlier Python-example definitions needed by a section, without executing examples. It returns only source spans within 1,800 body characters. A decision-tree heading triggers the disclosed prefix fallback. Otherwise, sections with their dependency closures are considered within the cap; the prefix is used if no source sections fit. If extraction would lose a high-confidence facet already credited to the prefix, that prefix is retained. The full contract is then checked against 9,000 characters; an overflow is rejected rather than silently truncated.

Notation. The implementation uses internal token sets for normalized query terms, technology anchors, operation hints, artifacts, constraints, failure cues, and visible-check cues. These correspond to the query schema $\psi(q)$ in Section 3. We use the paper notation throughout; F_s denotes

Table 11: Offline group-pool construction rules used by Algorithm 1.

Rule	Definition
TYPEDNEIGHBORHOOD	Collect positive incoming/outgoing dependency, workflow, artifact, visible-check, fallback, alternative, and semantic edges, in that priority order. Keep each neighbor's best edge; order by priority, descending weight, and stable IDs; retain at most two neighbors.
ENUMERATEGROUPS	Enumerate the singleton, lead–neighbor pairs, and the lead with two retained neighbors, subject to the three-member cap. Only triples require the second support to add a distinct role or an artifact/check facet beyond the lead and first support.
ASSIGNROLES	The lead is the anchor. Incoming/outgoing dependency gives prerequisite/dependent; workflow gives preprocessor/postprocessor; artifact gives parser/formatter. Visible-check gives checker; alternative/fallback gives fallback; semantic gives support.
EXTRACTGROUPFACETS	Extract facets from skill names, descriptions, source text, and declared public metadata. Lead facets are required; other member facets are optional. Negatives are separate; IDs, paths, and exclusivity fields do not create positive capabilities.
COMPATIBLE	Reject explicit incompatibility edges, member-negative versus positive-facet/name/ID conflicts, `exclusive_with` matches, or disjoint choices in a common declared exclusivity category. Different technology or format mentions alone do not imply incompatibility.
NONREDUNDANT / CANONICALINSERT	Apply the triple novelty rule above. Canonicalize by lead and sorted members and retain the higher-prior duplicate. Pairs have no additional general facet-novelty requirement.
UPDATEINDEX	Build member-to-group and positive-facet-to-group inverted indices after deduplication.
BUILDGROUPGRAPH	Lift atomic edges with their direction and relation type; add exact facet-similarity links. Symmetric incompatibility suppresses every positive relation for that pair. Positive relations must reach affinity 0.35; details follow below.

offline skill facets, while $F_s^{\text{vis}}(q)$ denotes facets of the presented name/body. Bundles record source spans, policy/configuration hashes, selected roles, and remaining debt so that this distinction can be inspected.

E Additional Experimental Details

E.1 Benchmark and Evaluation Protocol

The evaluation covers SkillsBench (Li et al., 2026c), ALFWorld (Shridhar et al., 2021), and Terminal-Bench 2. Table 1 compares skill-access methods across six backbones on SkillsBench, ALFWorld, and Terminal-Bench 2. Table 2 examines group selection through matched SkillsBench controls on 87 tasks.

Task coverage and environment requirements. The six-backbone comparison uses the full SkillsBench 1.1 suite (87 tasks), all 134 unseen ALFWorld `eval_out_of_distribution` tasks, and the full 89-task Terminal-Bench 2 suite. The GPT-5.4 SkillsBench control study also covers all 87 tasks. Each model–method cell uses three repeats. Tables 15 and 16 report the valid-run counts. The Vanilla Skills condition uses the skills supplied with each SkillsBench task. ALFWorld and Terminal-Bench 2 expose their full experimental libraries. These are the configurations of the reported runs. The current public SkillsBench runner mounts an externally supplied skill directory, so reproducing this Vanilla condition requires supplying the corresponding task-specific skills. The public release and its default task manifest are separate from the final evaluation summaries reported here.

All three benchmarks in the six-backbone comparison use 2,000-skill libraries, as does the SkillsBench control study. Smaller libraries are used only in the library-size analyses.

Table 12: Online selection and rendering rules used by Algorithm 2.

Rule	Definition
EXTRACTSCHEMA	Extract seven categories from the query and explicitly supplied public context using fixed lexical dictionaries. Skill metadata remains an offline retrieval input; it does not add query requirements.
HIGHCONFIDENCEFACETS	Use explicit canonical technologies, named API calls, recognized formats/extensions, artifact filenames, and stated constraints. An alias alone need not be high-confidence; a filename obligation is distinct from its format.
CANDIDATEGROUPS	Union query-facet postings with seed-member postings. Reject a group when its negatives conflict with high-confidence query facets, or its anchors conflict with query negatives.
TOPGROUPS	Rank by weighted group score plus prior; apply the adaptive cap, floor, and guaranteed minimum above. Ties use lead seed rank, group size, and stable group ID.
SELECTANCHOR	Add anchor correction. A group is admissible if its corrected score is at least 0.14, its lead is eligible, and its reserved set fits the budget. Prefer admissible groups with positive lead-anchor match; otherwise take the best admissible group. Return no anchor if none is admissible.
BESTSUPPORT	Require member compatibility, query eligibility, grounded evidence and marginal novelty as defined below, score at least 0.10, and reserved-set feasibility. Maximize support score with the group tie-breaker. Stop at three groups or no admissible option.
GROUPNEIGHBORS	Collect positive outgoing non-incompatibility neighbors of the plan. A group edge alone does not satisfy the grounded-support predicate.
INSERTRESERVED	Insert unique leads in plan order, then positive-seed incoming dependency/workflow members with prerequisite/preprocessor roles. Admission checks this entire reserved set for eligibility, compatibility, skill count, and rendered-context fit.
BESTSKILL	Among activated members of the plan, require eligibility, compatibility, positive marginal coverage/check/connectivity, score at least 0.10, and fit. Maximize score, then use seed rank and stable skill ID.
COVERAGEDEBT	Subtract the union of currently presented name/body facets from high-confidence query facets; do not credit metadata or omitted text.
BESTBACKFILL	Among activated skills outside B, require visible debt gain, eligibility, compatibility, and fit. Prioritize debt facets covered, then bottleneck score, seed rank, and skill ID. Add at most two; the ordinary skill-score threshold is not reapplied.
ANCHORPRUNE	If needed, promote the first plan group whose presented lead matches a high-confidence anchor. Retain only groups contributing a presented member.
FORMATCONTRACT	After source presentation, recompute debt and render START, SUPPORT, CHECK, AVOID, SKILLS, and DEBT. Enforce the final 9,000-character cap, including all fields and paths.

The implementation description in this appendix follows the public repository at commit `3c7de99`. Its configurable budgets and thresholds permit alternative settings; the reported experiments use the settings listed here.

Development and evaluation separation. The 24 scoring coefficients and four facet dictionaries were fixed using 10 development tasks selected from the official `tasks-extra` pool, disjoint from all 87 main SkillsBench tasks. The same tasks supply the 40 gate annotations. Thus end-to-end reward is evaluated separately from development, whereas gate coverage is a development-set mechanism diagnostic.

Metrics and task-level diagnostics. The task-level analyses in this appendix focus on SkillsBench because its tasks are paired with reusable skills and deterministic checks, making it possible to inspect retrieved skill coverage and matched trajectories. Each task is executed by the same downstream agent loop, with only the retrieved skill context changed across methods. We use reward as the

primary end-to-end metric and report token usage and runtime as efficiency metrics. Graph of Skills retrieves dependency-aware skill bundles, while GoSkills retrieves anchor-centered skill groups and expands support groups before rendering the final execution contract. The retrieval-gate and case-study tables below are therefore task-level SkillsBench analyses, not a separate re-aggregation of the ALFWorld results.

For retrieval evaluation, we define task-specific `must_have` skills and evaluate 40 annotated visible-requirement items per gate mode. A requirement-level pass means that the final presented context includes the required skill for that item. A partial result covers a related but incomplete required skill set, and a miss covers none. We report requirement-level pass, partial, miss, average must-hit rate, selected group count, and presented skill count.

We separate infrastructure failures from substantive agent failures. Runs with environment construction errors, Docker failures, or startup failures are tracked separately. If the agent has already entered a meaningful trajectory and then times out, we treat the timeout as a substantive execution failure. Invalid runs are not used as evidence for retrieval quality.

E.2 Agent-facing instructions in the evaluation harness

All methods share the same agent loop, task instruction, and environment; they differ only in the skill-access instructions and the retrieval command. On SkillsBench, every task instruction is preceded by the same one-sentence prefix, and each method provides its own `/root/AGENTS.md`. The prefix and the GoSkills instructions are reproduced verbatim below; Table 13 lists how the other arms differ.

Task prefix (all methods, SkillsBench)

```
Before writing or editing solution code, read /root/AGENTS.md if it exists. Follow its skill-access
↪  condition, inspect visible requirements/tests, preserve exact output paths, and prefer the
↪  shortest verifier-aligned solution.
```

`/root/AGENTS.md` for GoSkills (verbatim)

````
# Task Environment

This environment contains a prebuilt **GoSkills** group pool.

## Required First Step

Before writing any code, retrieve relevant skills by running a targeted query:

```bash
goskills-query "goal + artifact/format + operation/API + verifier-critical constraint"
```

This command is available on PATH. It queries a prebuilt group pool and returns
a bounded role-labeled contract and skills relevant to your task.

When writing the query, include only the retrieval-critical task facts that are actually known:

- concrete goal
- artifact, file format, or main object
- key operation, algorithm, API, or library
- verifier-critical constraint or invariant

Examples:

```text
update embedded xlsx in pptx preserve formulas
parallel tfidf search processpoolexecutor deterministic ranking
exact civ6 district adjacency calculator
```

Avoid vague queries such as `solve this task` or `help with benchmark`.
````

```
Read the START and DEBT fields. If no anchor/skills were selected, note the no-hit and continue
↪  without claiming skill use. Otherwise use only the presented skills and their stated roles.

## Reading the Output

Each returned skill includes a `Source:` path, e.g.:

```
Source: /opt/graphskills/skills/mesh-analysis/SKILL.md
```

The full skill directory (including `scripts/` and `references/`) is at that path.
For example: `/opt/graphskills/skills/mesh-analysis/scripts/mesh_tool.py`

Always check whether a ready-made script exists before implementing from scratch.

Before implementing, inspect the task requirements, tests, and verifier and identify the minimum
↪ acceptance requirements.

Priorities:

1. Take the shortest path to passing the verifier.
2. Pass only the verifier's minimum required behavior first.
3. Use the exact `Source:` path returned by retrieval. Do not reconstruct paths from the skill name
↪ or scan the whole skill library if a `Source:` path is already available.
4. Reuse or adapt retrieved scripts and interfaces when they directly fit the verifier target.
5. If a retrieved skill contains an authoritative calculator, validator, parser, or pack/unpack
↪ workflow, use that exact interface for the final output and final self-check.
6. Treat skills as a way to shrink the search space, not as permission to explore more
↪ implementation branches.
7. Avoid extra features, UI expansion, side outputs, or generalization unless explicitly required.

Workflow

1. Inspect the task and form a short retrieval query containing `goal + artifact/format +
↪ operation/API + verifier-critical constraint`
2. Run `goskills-query "<targeted query>"`
3. Record whether retrieval was a skill hit or no-hit
4. Inspect task requirements/tests/verifier and write down the minimum acceptance requirements
5. Read the returned skill bundle and use the exact `Source:` paths it provides
6. Check for scripts at the parent directory of each returned `Source:` path
7. Use or adapt those scripts only if they directly help satisfy the minimum requirements
8. If no returned script/interface directly fits, stay on the shortest no-frills path to verifier
↪ pass
9. Before finalizing, run one verifier-aligned self-check with the retrieved skill's authoritative
↪ interface when available

GoSkills uses the fixed START, SUPPORT, CHECK, AVOID, SKILLS, and DEBT fields. Respect the returned
↪ roles and report remaining coverage debt. Only the presented skills are selected; unpresented
↪ group members are not implied.
```

E.3 How retrieval queries are formed

The three benchmarks expose retrieval to the agent differently, but within a benchmark every retrieval-based method uses the same mechanism.

- **SkillsBench.** After reading the instructions above, the agent writes a short query and calls the method's retrieval command; in repeat 2 every GoSkills and Graph of Skills run issued exactly one query. In the protein case of Appendix G, the agent's query was `update xlsx spreadsheet formulas preserve formatting exact output path`.
- **ALFWorld.** The harness retrieves once from the initial observation and task goal. During the episode, the agent may emit `SkillRequest: GOS_RETRIEVE <goal or blocker>` for a new retrieval or `SkillRequest: READ_SKILL <name>` to read a skill; each request consumes one interaction step.

Table 13: Skill-access instructions per method on SkillsBench. Apart from the entries below, the GoSkills, Graph of Skills, and Vector Skills instructions are identical, including the query guidance, the priorities, and the workflow.

Method	Retrieval command	Described output	Reading rule
No Skills	None	None	"No external skill library is supplied."
Vanilla Skills	None; skill files are read on demand	Mounted skill directory (`/opt/skillsbench/skills`)	Use skills only when they directly help satisfy the verifier.
Vector Skills	`vectorskills-query`	Embedding-similarity results only	Report `SKILL_HIT` or `NO_SKILL_HIT`; use skills only as constraints; do not hide retrieval failures.
Graph of Skills	`graphskills-query`	"A ranked bundle of skills"	Report `SKILL_HIT` or `NO_SKILL_HIT`; use skills only as constraints.
GoSkills	`goskills-query`	"A bounded role-labeled contract"	Read START and DEBT; use only the presented skills and their stated roles.

- **Terminal-Bench 2.** The harness retrieves with the full task instruction before the first turn; the agent can issue follow-up queries through a `retrieve_skills` tool.

ALFWorld system prompt and skill-access message (GoSkills)

```
Interact with a household to solve a task. Imagine you are an intelligent agent in a household
↪  environment and your target is to perform actions to complete the task goal. At the beginning of
↪  your interactions, you will be given the detailed description of the current environment and
↪  your goal to accomplish.
For each turn, think briefly and then choose exactly one of these two outputs:
1. `Thought: ...` followed by `Action: ...` for a normal environment action.
2. `Thought: ...` followed by `SkillRequest: ...` if you want extra skill help before acting.

Use `SkillRequest` when it directly helps the next step, especially if you are blocked, the action
↪  syntax is unclear, the retrieved skill names look unrelated to the current blocker, or 1-2
↪  actions already failed. In GoS mode, prefer `SkillRequest: GOS_RETRIEVE ...` when you need a
↪  better-targeted skill search. In vector mode, prefer `SkillRequest: VECTOR_RETRIEVE ...` when
↪  you need a better-targeted vector-only search. Use `READ_SKILL` only when you already know which
↪  exact skill is worth reading. Do not output both `Action:` and `SkillRequest:` in the same turn.

The available actions are:
1. go to {recep}
2. take {obj} from {recep}
3. move {obj} to {recep}
4. open {recep}
5. close {recep}
6. use {obj}
7. clean {obj} with {recep}
8. heat {obj} with {recep}
9. cool {obj} with {recep}
where {obj} and {recep} correspond to objects and receptacles.
After your each turn, the environment will give you immediate feedback based on which you plan your
↪  next few steps. if the envrionment output "Nothing happened", that means the previous action is
↪  invalid and you should try more options.

Your response should use one of the following formats:

Thought: <your thoughts>
Action: <your next action>

or
```

```
Thought: <your thoughts>
SkillRequest: <your tool-style request>

[user]
GoSkills provides an execution contract from the current visible query. Follow applicable skills
↪  conditionally. For a focused new retrieval use `Thought: ...` then `SkillRequest: GOS_RETRIEVE
↪  <current visible goal or blocker>`. To read a named skill use `SkillRequest: READ_SKILL <exact
↪  skill name>`. Each request consumes one interaction step; do not also emit Action.

[user]
You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a
↪  drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a
↪  laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.
Your task is to: examine the alarmclock with the desklamp.
```

The last `[user]` message is the first observation of ALFWorld task 000.

E.4 A rendered contract from the evaluation

The box below is the contract that GOSKILLS returned to the agent for `protein-expression-analysis` in repeat 2 (Case 1 of Appendix G). All field text is verbatim; only the skill payloads are replaced by their lengths. The anchor (Excel Automation) documents cloud spreadsheet APIs; the verifier-critical recalculation step comes from the `xlsx` support skill.

Rendered contract for `protein-expression-analysis` (payloads elided)

```
START
Use Excel Automation first because its group matches the visible task evidence.
Inspect its source path before writing new code.

SUPPORT
Use these skills only for the stated roles:
- xlsx: support lead - supports the selected plan or an explicit requirement.
- recipe-log-deal-update: support lead - supports the selected plan or an explicit requirement.
- Excel Analysis: dependent - supports the selected plan or an explicit requirement.

CHECK
Before finalizing, verify these visible requirements:
- artifact:xlsx

AVOID
Do not follow these incompatible interpretations:
None identified from explicit applicability cues.

SKILLS
Skill: Excel Automation
Source: /opt/graphskills/skills/excel-automation/SKILL.md
[source-preserving excerpt of 1,579 characters omitted]

Skill: xlsx
Source: /opt/graphskills/skills/xlsx/SKILL.md
[source-preserving excerpt of 1,724 characters omitted]

Skill: recipe-log-deal-update
Source: /opt/graphskills/skills/recipe-log-deal-update/SKILL.md
[source-preserving excerpt of 625 characters omitted]

Skill: Excel Analysis
Source: /opt/graphskills/skills/excel-analysis/SKILL.md
[source-preserving excerpt of 1,616 characters omitted]

DEBT
None.
```

E.5 SkillsBench Control Study

We evaluate group selection on all 87 SkillsBench 1.1 tasks with GPT-5.4 and three repeats per cell. Table 2 is the primary comparison for assessing group selection. All arms share the atomic candidate universe, query schema, skill payloads, agent, and environment, with at most four payloads, 1,800 characters per payload, and a 9,000-character context guard. Graph + Fixed Contract keeps the structural baseline's payloads fixed and transfers both contract fields and reading instructions. The post-hoc (PH) controls assign START, SUPPORT, CHECK, and AVOID roles to the top retrieved skills after retrieval, using a rule-based annotator (PH-Rule) or an LLM-based annotator (PH-LLM), and perform requirement backfill within the shared pre-group candidate universe; neither constructs groups or traverses a group graph. Each arm retains the reading instructions for its own skill-block format.

GOSKILLS improves reward over Graph of Skills by 3.7 percentage points (35.7 to 39.4) and uses 1.3 fewer turns (15.8 to 14.5). Relative to the fixed-contract Graph of Skills control, the reward difference is +2.2 points (paired 95% CI $[-3.9, 8.8]$), and the turn difference is -0.9 (95% CI $[-2.0, 0.2]$). Hosted-API timing remains sensitive to provider load. Tool-call and repair-attempt counts were not separately measured.

The post-hoc controls reach mean rewards of 36.7 and 37.4, compared with 39.4 for full GOSKILLS. Group expansion can expose visible-check skills that rank low or are absent from seed retrieval, enlarging the activated candidate set before bottlenecking. The gate ordering (0.81/0.83 for PH controls versus 1.00 for GOSKILLS) is consistent with that mechanism, but the gate's use in development limits generalization of the coverage result. These controls evaluate the combined selection and rendering pipeline; the point estimates indicate a benefit from group selection.

Contract rendering and payload selection. Rendering the same Graph of Skills payloads with the fixed contract raises the observed mean reward by 1.5 points. Full GOSKILLS adds a further 2.2 points relative to this control.

E.6 Robustness to Metadata and Atomic-Graph Corruption

The atomic graph is produced by the Graph of Skills benchmark's deterministic construction script, without an LLM; GOSKILLS adds the group layer. Group-level relations also use metadata facets, which are the dominant construction evidence on the evaluated corpus. Consequently, incomplete atomic edges and missing metadata need not have the same effect. Table 14 gives the complete corruption sweep under GPT-5.4. Perturbations precede indexing and group construction, and all arms share the perturbed library with no hyperparameter retuning. Reward is measured on all 87 SkillsBench tasks. Reported standard deviations are over three corruption seeds, not estimates of repeated-run variation for the clean aggregate tables. Must-hit uses the 40 annotated development-gate items. The comparator is Graph + Fixed Contract, which matches the contract while keeping the structural baseline payloads fixed.

Table 14: Robustness to imperfect documentation and atomic graph structure. Reward is in percent; $\pm$ denotes sample s.d. across three library-seed means where available, after averaging three agent repeats per seed. Clean and deterministic raw-document conditions have no corruption-seed s.d. Hit is instruction-auto development-gate must-hit. Δ is the reward gap in percentage points; retention is GOSKILLS reward relative to its clean value.

	Fixed contract		GOSKILLS			
Condition	**Reward**	**Hit**	**Reward**	**Hit**	Δ	**Retention**
Clean	37.2	0.84	39.4	1.00	+2.2	100.0%
25% metadata blanking	36.4 ± 0.53	0.80	37.9 ± 0.73	0.95	+1.5	96.2%
50% metadata blanking	35.6 ± 0.78	0.74	36.2 ± 0.95	0.88	+0.6	91.9%
25% typed-edge removal	36.9 ± 0.37	0.83	39.0 ± 0.64	0.98	+2.1	99.0%
50% typed-edge removal	36.7 ± 0.65	0.82	38.5 ± 0.86	0.96	+1.8	97.7%
25% metadata + 25% edge	36.1 ± 0.80	0.78	37.4 ± 0.82	0.93	+1.3	94.9%
Raw name and description only	35.3	0.74	36.0	0.88	+0.7	91.4%
+20% lexical distractors	36.8 ± 0.63	0.82	38.7 ± 0.55	0.98	+1.9	98.2%

The positive margin is preserved in every tested condition. Metadata loss has a larger effect than deletion of atomic typed edges: at 50% corruption the reward reductions are 3.2 and 0.9 points, respectively. This distinction matters because the group graph also derives relations from metadata facets; deleting atomic edges is not equivalent to ablating the group graph. Raw name-and-description inputs and lexically similar distractors test less curated inputs, but do not establish robustness to arbitrary documentation failures or unseen domains.

E.7 Transfer Scope and Efficiency Accounting

Terminal-Bench 2 protocol. The complete 89-task suite is evaluated across all six backbones with three repeats per cell (Table 1). The 2,000-skill library is held fixed across methods, with no task-specific additions, removals, or edits. Budgets, agent settings, and environments are fixed across methods within this study. The paper's agent loop is retained instead of the benchmark-specific leaderboard harness, precluding direct leaderboard comparison. The task suite differs from SkillsBench, while the library size remains 2,000. Relative to Graph of Skills, accuracy gains range from 6.7 to 14.6 percentage points on Terminal-Bench 2, while reward gains range from 1.6 to 4.8 points on SkillsBench. These cross-benchmark differences do not isolate the effect of domain shift. No paired significance claim is made.

ALFWorld gains. Relative to No Skills, the Gemini 3 Pro and GPT-5.4 gains remove about 71% and 73% of the remaining error, respectively. These are descriptive normalized gains. ALFWorld exposes fewer technical requirements and lacks the same visible deterministic-check structure as SkillsBench.

Which tasks benefit. The 87-task matched analysis has 19 GoSkills wins, 11 Graph of Skills wins, and 57 ties (Table 23). Ties are the most common outcome. The cases in Appendix G show a win where the presented excerpt supplies a verifier-critical step, a tie where both methods share the entry point, and a loss on a solver-level detail. Long setup chains remain a difficult case.

Turns, tokens, and hosted-API time. A turn is one model-invocation–action cycle of the fixed agent loop. Counts are insensitive to provider queueing and batching but remain model-dependent. In the 87-task control study, turns fall from 15.8 to 14.5 versus Graph of Skills and from 15.4 to 14.5 versus the fixed-contract control. Input tokens accumulate over the entire agent loop. In the ablation, removing role labels decreases input tokens from 156,696 to 153,558 and increases agent time from 187.3s to 190.9s. Adding the contract to Graph of Skills increases input tokens from 136,054 to 144,517 and decreases agent time from 192.9s to 190.6s. Higher cumulative token usage can coexist with fewer turns because the context per invocation can vary. The observed mean turn counts are consistent with reduced organizational effort; hosted-provider effects can also contribute to runtime differences.

Offline group construction. At library sizes of 200, 500, 1,000, and 2,000 skills, measured build times are 1.8, 5.9, 14.7, and 38.2 seconds, respectively. Peak heap is 92, 148, 236, and 412 MB, and bundle overhead is 1.4, 3.6, 7.5, and 15.6 MB. The scaling fit estimates a time exponent of 1.30 (interval $[1.21, 1.40]$) and a 2,000-skill extrapolation of 36.2 seconds (interval $[30.4, 43.1]$).

Uncertainty and provenance. Run-provenance counts in Table 15 cover all three benchmarks. Appendix E.17 reports sample standard deviations across the three repeats for the main results and control/ablation study. The reported corruption standard deviations measure variation over three corruption seeds, not clean-run stochasticity. Paired task bootstrap intervals quantify variability under task resampling, conditional on observed repeats. Three repeats provide limited information about run-to-run variation.

E.8 Run provenance and aggregation

Table 15 reports the run provenance used for the aggregate results across all three benchmarks in Table 1. We separate infrastructure failures from task-level failures. Infrastructure failures include API outages, environment crashes, or logging failures that prevent the benchmark evaluator from producing a valid task outcome. These runs are excluded from aggregate reward, token, and runtime computation and are not imputed. In contrast, agent timeouts are retained as valid task-level outcomes

when the evaluator returns a task result; they are counted as task failures for reward and included in token/runtime accounting up to the timeout cap.

For each model–method–benchmark cell, reward is first averaged over valid task outcomes within each repeat and then averaged over the three repeats. Token usage reports mean cumulative input tokens over the same valid task runs, and runtime reports mean agent-only task-processing time, excluding environment setup. The table aggregates provenance over the six evaluated backbones for compactness; the same accounting rule is applied to every model–method–benchmark cell.

For space, Table 15 aggregates provenance over backbones. Table 21 reports task-level paired intervals.

Table 15: Run provenance for the six-backbone comparison on SkillsBench, ALFWorld, and Terminal-Bench 2. Attempted task-runs are aggregated over the six evaluated backbones and three repeats. Valid reward denotes task-runs with evaluator-produced outcomes used in Table 1. Infra. failures are excluded and not imputed. Timeout failures are a subset of valid reward task-runs and are counted as task failures.

Benchmark	Method	Tasks / model	Backbones	Attempted	Valid reward	Infra. fail.	Timeout fail.
SkillsBench	No Skills	87	6	1566	1558	8	110
SkillsBench	Vanilla Skills	87	6	1566	1555	11	94
SkillsBench	Vector Skills	87	6	1566	1557	9	104
SkillsBench	Graph of Skills	87	6	1566	1556	10	114
SkillsBench	GoSkills	87	6	1566	1559	7	98
ALFWorld	No Skills	134	6	2412	2412	0	660
ALFWorld	Vanilla Skills	134	6	2412	2412	0	493
ALFWorld	Vector Skills	134	6	2412	2412	0	440
ALFWorld	Graph of Skills	134	6	2412	2412	0	403
ALFWorld	GoSkills	134	6	2412	2412	0	335
Terminal-Bench 2	No Skills	89	6	1602	1602	0	318
Terminal-Bench 2	Vanilla Skills	89	6	1602	1602	0	412
Terminal-Bench 2	Vector Skills	89	6	1602	1602	0	359
Terminal-Bench 2	Graph of Skills	89	6	1602	1602	0	338
Terminal-Bench 2	GoSkills	89	6	1602	1602	0	346

The control study uses the same accounting as the SkillsBench comparison (Table 16). ALFWorld and Terminal-Bench 2 rerun initial environment failures and report the final run status. ALFWorld timeouts are episodes reaching the step limit.

E.9 Retrieval-gate annotation protocol

The retrieval gate is designed as a visible-requirement coverage diagnostic for SkillsBench on the selected 10-task `tasks-extra` development subset, not as held-out coverage or a separate end-to-end benchmark. These cases also determined the scoring coefficients and facet dictionaries. Table 17 shows how we evaluate whether the final presented context contains the task-specific `must_have` skills needed to satisfy visible task requirements before agent execution.

The `must_have` annotations were created from task prompts, public task files, skill names, skill metadata, and skill payload descriptions. They exclude hidden tests, evaluator internals, private oracle information, previous failure traces, and any artifacts produced during agent execution. Each annotation item links one visible requirement to one or more required skills. Examples include explicit file formats, named APIs, public checks, deterministic output requirements, or formal proof obligations.

Table 16: Run accounting for the GPT-5.4 SkillsBench control study. Timeouts remain valid zero-reward runs.

Method	Attempted	Valid reward	Infra. fail.	Timeout
Graph of Skills	261	260	1	21
Graph of Skills + GoSkills Contract	261	259	2	20
PH-Rule + backfill	261	259	2	21
PH-LLM + backfill	261	259	2	20
GoSkills	261	260	1	18
w/o Anchor Selection	261	260	1	19
w/o Group Expansion	261	260	1	19
w/o Group Graph	261	260	1	19
w/o Role Labels	261	260	1	19
w/o Coverage Backfill	261	260	1	20
w/o Avoid Field	261	260	1	19
Retrieved Skills Only	261	259	2	21

Annotations were produced before final retrieval-gate evaluation. The annotators did not inspect final GoSkills outputs when assigning `must_have` labels. Disagreements were resolved by discussion using only visible task materials and the skill-library metadata. The final annotation set contains 40 visible-requirement items per gate mode, covering **10** SkillsBench gate tasks. The same annotation set is used for all compared methods and ablations.

The gate evaluation is separated from the method features. GoSkills uses deterministic query facets, normalized skill metadata, typed graph relations, and visible-check cues for retrieval and bottlenecking. The gate labels are task-level evaluator annotations used only after retrieval to check coverage. Although both the method and the gate rely on visible task information, the gate does not use GoSkills group scores, selected groups, coverage-debt state, or contract fields as labels. This avoids evaluating the method with labels derived from its own outputs.

Table 17: Retrieval-gate annotation protocol. The same visible-requirement annotations are used for all methods evaluated under the gate.

Question	Protocol
Who annotated `must_have` skills?	Two authors independently annotated visible requirements from task prompts, public files, skill names, skill metadata, and payload descriptions. Disagreements were resolved by discussion using only visible task materials and skill-library metadata.
Blind to GoSkills outputs?	Annotators did not inspect final GoSkills retrieved contexts, selected groups, or rendered contracts when assigning `must_have` labels. The labels were fixed before the final retrieval-gate evaluation.
Same facets as the method?	No. GoSkills uses deterministic query/schema facets for retrieval and bottlenecking, while the gate uses fixed task-level annotations for post-hoc coverage evaluation. Both are restricted to visible task information, but gate labels are not derived from GoSkills selected groups, scores, coverage-debt state, or rendered contracts.
Hidden information excluded?	Yes. Hidden tests, evaluator internals, private oracle information, previous failure traces, and execution-time artifacts are excluded. The annotations use only information available before agent execution.
Coverage of 40 items?	The final gate contains 40 visible-requirement items per mode across **10** SkillsBench gate tasks. Each item maps one visible requirement, such as an explicit file format, named API, public check, deterministic output constraint, or proof obligation, to one or more required skills.
Other methods evaluated?	Yes. The same gate labels are applied to Graph of Skills, the post-hoc controls, GoSkills, and its ablations using each method's final presented context before agent execution.

E.10 Retrieval Gate Details

To isolate retrieval quality from downstream execution, we check whether the presented context contains all annotated `must_have` skills before agent execution. The gate contains 40 visible-requirement items per mode on the 10-task `tasks-extra` development subset, so coverage is a mechanism diagnostic rather than held-out evidence. Table 18 shows that GOSKILLS passes all 40 requirement items under each mode, achieving a 1.00 must-hit rate with fewer than four presented skills on average. Compared with the no-backfill ablation in Table 2, this indicates that coverage-safe backfill helps preserve task-critical skills under a strict bottleneck.

Table 18: Retrieval-gate results over 40 annotated visible-requirement items per mode on the 10-task `tasks-extra` development subset. These results are mechanism diagnostics, not held-out coverage estimates. Req. P, Req. Par., and Req. M denote requirement-level pass, partial, and miss. Hit denotes the average must-hit rate, and Skills denotes the average number of presented skill payloads. Groups denotes the mean number of selected groups.

Mode	Req. P	Req. Par.	Req. M	Hit ↑	Skills ↓	Groups
`instruction_auto`	40	0	0	1.00	3.10	2.30
`critical_override`	40	0	0	1.00	2.90	2.20

Table 19 shows the improvement from the initial retrieval configuration to the final gate over the same 40 annotated visible-requirement items per mode. The average must-hit rate improves from 0.73 to 1.00 in both modes.

Table 19: Initial and final retrieval-gate results on the same 10 development tasks.

Stage	Mode	Req. P	Req. Par.	Miss	Must-hit
Initial gate	`instruction_auto`	27	4	9	0.73
Initial gate	`critical_override`	28	2	10	0.73
Final gate	`instruction_auto`	40	0	0	1.00
Final gate	`critical_override`	40	0	0	1.00

E.11 Task-Level Retrieval Examples

Table 20 presents retrieval examples from the main SkillsBench evaluation tasks, produced after the scoring coefficients and facet dictionaries were fixed. These cases were not used for tuning or development-gate annotations. They illustrate how GOSKILLS combines a primary skill with supporting skills that cover artifacts, workflow dependencies, or visible-check constraints.

Table 20: Illustrative retrieval outputs on main evaluation tasks under `instruction_auto`, after parameter freezing and separate from the development gate. These gate-mode queries are generated automatically and can differ from the agent-written queries of the end-to-end runs, so the presented skills can differ from those in Appendix G.

Task	Primary skill	Presented skills
`invoice-fraud-detection`	`fuzzy-match`	`fuzzy-match`, `pdf-reading`, `xlsx`
`gravitational-wave-detection`	`conditioning`	`conditioning`, `matched-filtering`, `silence-detector`
`threejs-structure-parser`	`threejs`	`threejs`, `obj-exporter`, `discover-important-function`
`lean4-proof`	`lean4-theorem-proving`	`lean4-theorem-proving`, `lean4-memories`

E.12 Task-level paired bootstrap confidence intervals

To quantify the stability of the SkillsBench comparisons, we compute task-level paired bootstrap confidence intervals for reward and runtime deltas. For each task and method, we first average

repeated valid runs within the task. We then form paired task-level differences between GOSKILLS and each baseline and resample tasks with replacement for 10,000 bootstrap samples. Only tasks with valid results for both methods in a given comparison are included in that paired comparison. All four comparisons reported here contain 87 paired tasks. We report the mean paired delta and the percentile 95% confidence interval. These intervals quantify variability under task resampling, conditional on the observed repeats, rather than run-to-run stochasticity. Paired intervals against the fixed-contract control are reported in Appendix E.5.

Table 21 reports a +3.7-point paired reward difference versus Graph of Skills (95% CI $[-2.6, 10.4]$) and a -5.6s runtime difference (95% CI $[-24.1, 14.7]$).

We emphasize that the bootstrap is paired at the task level: each resampled unit contains both the GOSKILLS result and the corresponding baseline result for the same task. This controls for task difficulty and directly estimates the stability of method-level differences rather than comparing independent aggregate means.

Table 21: Paired GPT-5.4 SkillsBench differences (GOSKILLS minus baseline) and percentile 95% confidence intervals from 10,000 task-level bootstrap resamples. Runtime differences and intervals are in seconds.

Baseline	**Paired N**	**Δ Reward (pp) $\uparrow$**	**95% CI**	**Δ Runtime $\downarrow$**	**95% CI**
Graph of Skills	87	+3.7	[−2.6, +10.4]	−5.6	[−24.1, +14.7]
Vector Skills	87	+5.4	[−1.2, +12.5]	+1.5	[−18.2, +22.4]
Vanilla Skills	87	+10.5	[+3.5, +17.9]	+28.1	[+8.1, +50.2]
No Skills	87	+14.1	[+6.9, +21.6]	−1.3	[−21.4, +20.1]

E.13 Baseline specification and accounting

Table 22 summarizes the baseline interfaces used in the aggregate experiments. The goal is to make the comparison about the organization of retrieved context rather than about different downstream agents, execution loops, or skill implementations. All methods use the same downstream agent loop, execution environment, task prompts, and skill payloads. For retrieved-skill methods, we use the same exposed-payload cap of four atomic skill payloads and the same rendered-context guard of 9,000 characters. Thus, Vector Skills, Graph of Skills, and GOSKILLS are compared under the same final payload budget, while Vanilla Skills uses task-provided skills in SkillsBench.

The main difference among the retrieved-skill methods is the interface used to construct and render the skill context. Vector Skills exposes a flat semantic top-k list. Graph of Skills hydrates a dependency-aware bundle from the typed skill graph. GOSKILLS first selects an anchor/support group plan, bottlenecks the selected groups into the same four-payload budget, applies coverage-safe backfill when needed, and renders the result as a role-labeled execution contract. Therefore, differences between Graph of Skills and GOSKILLS should be interpreted as differences between a hydrated structural bundle and the full group-structured retrieval-and-rendering interface, rather than as changes to the downstream model or execution environment.

For accounting, reward is averaged over tasks within each run and then averaged across runs. Token usage reports mean cumulative input tokens, including the task prompt, the method-specific skill context, and the downstream agent prompt. Runtime reports mean agent-only task-processing time and excludes environment setup. Infrastructure failures are tracked separately from valid task outcomes, as described in Appendix E.1.

E.14 Matched Paired Results

Table 23 reports a matched SkillsBench subset in which Graph of Skills and GOSKILLS runs are aligned at the task and slice level. This subset is narrower than the full benchmark–model aggregation in Table 1; it is used to inspect matched outcomes and run-level reliability rather than replace the primary aggregate comparison.

In both subsets, ties remain the most common outcome, indicating that GOSKILLS often preserves downstream task success while changing the retrieval unit from atomic skills or post-hoc bundles to

Table 22: Baseline specification for aggregate experiments. All retrieved-skill methods use the same exposed-payload cap and the same payload truncation guard. GOSKILLS differs by selecting and rendering a role-labeled group plan before exposing the final atomic skill payloads.

Method	Exposed context	Budget / truncation	Prompt wrapper	Accounting
No Skills	No retrieved skill payloads.	No retrieval budget; no skill-context block.	Same downstream agent prompt with the skill block omitted.	Same reward, token, and runtime accounting.
Vanilla Skills	Task-provided skills on SkillsBench; the benchmark's full library on ALFWorld and Terminal-Bench 2.	No retrieval top-k or graph hydration.	Available skill instructions supplied directly to the agent.	Same accounting.
Vector Skills	Flat semantic top-k atomic skills.	Top-$k = 4$ exposed payloads; each payload uses the same truncation guard as other retrieved baselines; max rendered skill context is 9,000 characters.	Generic retrieved-skill block. No START, SUPPORT, CHECK, AVOID, or debt fields.	Same accounting.
Graph of Skills	Hydrated dependency-aware skill bundle from the typed skill graph.	At most 4 exposed payloads after graph hydration; same payload truncation guard; max rendered skill context is 9,000 characters.	Graph-hydrated skill block. Dependency structure is available through the bundle, but no explicit role-labeled execution contract is rendered.	Same accounting.
GOSKILLS	Anchor/support group plan rendered as a role-labeled execution contract plus final atomic payloads.	At most 3 selected groups internally; at most 4 exposed payloads after bottlenecking/backfill; same payload truncation guard; max rendered skill context is 9,000 characters.	Fixed START/SUPPORT/CHECK/AVOID/debt contract with the same downstream agent loop and execution environment.	Same accounting.

role-aware skill groups. At the same time, GOSKILLS obtains more wins than Graph of Skills in both subsets and achieves a higher average reward in each. In the completed matched subset, GOSKILLS improves average reward from 0.357 to 0.394, with 19 wins, 11 Graph of Skills wins, and 57 ties.

Table 23: Paired SkillsBench outcomes under GPT-5.4. The completed subset covers all 87 tasks over three repeats; the per-task snapshot uses repeat 1 only. Errors report infrastructure failures and timeouts (Graph of Skills/GOSKILLS).

Slice	Pairs	Graph W	GOSKILLS W	Tie	Graph R	GOSKILLS R	Errors
Completed matched subset	87	11	19	57	0.357	0.394	22/19
Per-task snapshot subset	40	4	10	26	0.429	0.483	3/2

E.15 Token and Runtime Analysis

Table 24 reports token and runtime statistics for the matched slices. GOSKILLS uses more input and total tokens and less wall-clock and agent-only time in each slice. On the full matched set, agent time falls from 192.9s to 187.3s while input tokens increase from 136,054 to 156,696.

Input tokens are provider-reported prompt/input tokens accumulated over recorded downstream-agent API responses throughout each task episode. Total tokens denotes the sum of these input tokens

and the corresponding completion/output tokens. The table reports means over included task runs. Infrastructure failures are excluded, while timeouts with valid evaluator outcomes remain included.

Table 24: Token and runtime comparison for matched trajectory slices. Lower values are better for all four metrics; bold denotes the lower value within each slice.

Slice	Method	Runs	Input tokens	Total tokens	Wall time (s)	Agent time (s)
Completed matched	Graph of Skills	260/261	**136,054**	**148,937**	236.8	192.9
Completed matched	GOSKILLS	260/261	156,696	171,284	**230.9**	**187.3**
Per-task snapshot	Graph of Skills	40/40	**129,806**	**141,877**	229.3	186.0
Per-task snapshot	GOSKILLS	40/40	149,412	163,329	**223.4**	**180.7**

E.16 Implementation Settings

Unless otherwise stated, GOSKILLS uses four seed skills for group activation, selects at most three groups, presents at most four atomic skills to the downstream agent, caps each hydrated skill payload at 1,800 characters, and caps the full rendered skill context at 9,000 characters. The scoring weights for group ranking, support expansion, and bottleneck selection are fixed across all benchmarks, models, and tasks. The downstream agent loop, execution environment, and skill payloads are kept unchanged across methods; only the retrieved skill context differs.

E.17 Variation Across Repeated Runs

Tables 25, 27, and 28 report sample standard deviations (SDs) for every benchmark–backbone–method configuration in Table 1; Table 26 covers all controls and ablations in Table 2. For each metric, x_r is its aggregate within repeat r, and $s = \sqrt{\sum_{r=1}^{3}(x_r - \bar{x})^2/2}$ is the sample SD across $n = 3$ repeats. Each repeat follows the valid-run and timeout accounting in Appendix E.8. These SDs describe variation across repeat-level aggregates, unlike the corruption-seed SDs in Table 14 and the task-resampling intervals in Table 21. Three repeats give a limited estimate of run-to-run variability; these SDs are not standard errors or confidence intervals. The corresponding means remain in the main tables.

F Failure and Error Analysis

Error taxonomy. We separate failures caused by the retrieval interface from failures caused by downstream execution or infrastructure. This distinction is important because GOSKILLS only changes the retrieved context. It does not execute skills, bind arguments, repair code after a failure, or inspect hidden evaluator state. Table 29 summarizes the categories used in our failure analysis.

Retrieval-side failures. In GOSKILLS, a retrieval miss can occur before group selection if the seed evidence fails to activate the relevant group, or after group selection if bottlenecking drops a required support skill. The retrieval-gate comparison in Table 19 shows this distinction empirically: the initial configuration produced misses and partial results, while the final gate reaches 40/40 requirement-level pass in both modes. We therefore treat coverage debt as an implementation check, not as a proof that execution will succeed. It only records whether high-confidence visible requirements remain uncovered by the final presented context.

Execution failures after valid retrieval. Some failures remain even when the retrieved context is plausible. In `energy-market-pricing` (Appendix G, Case 4), both runs complete with closely matching formulations, and the GOSKILLS run fails one optimality test at a 0.01% tolerance. Such outcomes should not be collapsed into infrastructure noise: a valid start and relevant skills do not guarantee correct multi-step execution. For GOSKILLS, the limitation is that group structure can make the entry point and support roles explicit, but it cannot force the downstream model to perform the right multi-step execution.

Efficiency failures. The group contract can add useful structure, but it is not free. On both matched slices in Table 24, GOSKILLS uses more tokens on average while reducing agent-only runtime.

Table 25: SkillsBench sample SDs across three repeats for Table 1. Reward is in pp, tokens are cumulative input tokens, and time is agent-only seconds.

Method	Reward SD	Token SD	Time SD	Method	Reward SD	Token SD	Time SD
GPT-5.4				**Gemini 3 Pro**			
No Skills	1.50	6,586	13.4	No Skills	1.13	8,485	11.4
Vanilla Skills	1.72	7,154	8.8	Vanilla Skills	1.88	7,406	9.7
Vector Skills	1.73	8,136	12.6	Vector Skills	2.30	10,595	13.1
Graph of Skills	1.92	6,750	14.5	Graph of Skills	2.65	9,460	11.8
GoSkills	2.19	8,885	11.9	GoSkills	2.48	10,552	12.0
MiniMax M2.7				**Claude Sonnet 4.5**			
No Skills	1.37	27,703	39.4	No Skills	2.11	22,103	23.6
Vanilla Skills	1.68	24,568	32.6	Vanilla Skills	2.37	27,043	20.3
Vector Skills	1.86	25,275	35.7	Vector Skills	2.75	28,759	24.1
Graph of Skills	1.80	30,061	43.0	Graph of Skills	1.73	26,912	22.1
GoSkills	1.65	31,343	38.9	GoSkills	2.56	31,548	22.3
Qwen 3.5				**Kimi K2.5**			
No Skills	2.15	24,684	46.9	No Skills	1.95	22,672	30.7
Vanilla Skills	2.37	27,068	44.2	Vanilla Skills	2.72	27,167	27.5
Vector Skills	2.57	30,649	34.3	Vector Skills	2.36	26,904	34.8
Graph of Skills	2.44	23,024	48.9	Graph of Skills	2.25	24,976	36.2
GoSkills	2.05	28,810	41.7	GoSkills	2.45	29,971	33.0

Table 26: Sample SDs across three repeats for the GPT-5.4 SkillsBench controls and ablations in Table 2. Reward SD is in pp, token SD is in cumulative input tokens, time SD is in agent-only seconds, and turn SD is in model-invocation–action cycles. Must-hit is reported separately as a development-gate diagnostic; these SDs cover the repeated end-to-end metrics.

Method	Reward SD	Token SD	Time SD	Turn SD
Graph of Skills	1.92	6,750	14.5	0.38
Graph of Skills + GoSkills Contract	1.70	8,448	13.1	0.60
PH-Rule + backfill	2.07	8,200	11.9	0.71
PH-LLM + backfill	2.29	7,922	11.4	0.55
GoSkills	2.19	8,885	11.9	0.64
w/o Anchor Selection	2.04	8,829	13.6	0.58
w/o Group Expansion	2.38	8,053	11.9	0.47
w/o Group Graph	2.01	7,917	11.0	0.62
w/o Role Labels	1.39	7,813	10.8	0.74
w/o Coverage Backfill	2.11	7,155	12.6	0.77
w/o Avoid Field	1.81	8,149	11.7	0.45
Retrieved Skills Only	1.89	7,419	11.6	0.84

The task-level pattern is also mixed: among the cases in Appendix G, GoSkills uses fewer input tokens on `protein-expression-analysis`, `3d-scan-calc`, and `energy-market-pricing` but more on `threejs-structure-parser`. This supports the scoped interpretation used in the main paper: group-structured retrieval is most useful when the task has a compact anchor/support/check decomposition, and less reliable when the dominant difficulty is broad setup or environment repair.

Infrastructure failures. We retain completed episodes as substantive outcomes, including reward-0 failures. Episodes with container-startup failures, dependency failures, or unavailable evaluator outcomes are separated as infrastructure evidence. Agent execution timeouts are retained as task failures, including zero-reward SkillsBench timeouts and ALFWorld step-limit failures. This policy prevents setup and reward-artifact failures from being misread as retrieval misses while still counting completed failed attempts as real downstream outcomes.

Table 27: ALFWorld sample SDs across three repeats for Table 1. Reward is in pp, tokens are cumulative input tokens, and time is agent-only seconds.

Method	Reward SD	Token SD	Time SD	Method	Reward SD	Token SD	Time SD
GPT-5.4				**Gemini 3 Pro**			
No Skills	2.69	1,060	2.8	No Skills	2.99	1,161	2.3
Vanilla Skills	1.97	85,239	6.1	Vanilla Skills	1.88	70,393	4.6
Vector Skills	2.59	2,501	3.3	Vector Skills	2.24	2,272	3.2
Graph of Skills	3.25	2,558	3.0	Graph of Skills	1.97	2,458	2.4
GoSkills	1.29	2,123	3.1	GoSkills	2.24	2,507	2.2
MiniMax M2.7				**Claude Sonnet 4.5**			
No Skills	2.40	2,810	7.5	No Skills	2.28	1,072	2.7
Vanilla Skills	3.25	135,903	9.2	Vanilla Skills	1.55	84,297	4.3
Vector Skills	1.88	3,712	6.0	Vector Skills	2.40	2,179	3.6
Graph of Skills	2.69	3,514	6.5	Graph of Skills	2.59	2,160	4.2
GoSkills	2.62	3,153	5.3	GoSkills	2.99	1,976	2.9
Qwen 3.5				**Kimi K2.5**			
No Skills	3.36	2,409	8.5	No Skills	2.62	1,906	6.4
Vanilla Skills	2.28	112,684	18.9	Vanilla Skills	2.83	90,812	15.0
Vector Skills	2.83	3,981	9.8	Vector Skills	3.11	3,124	6.9
Graph of Skills	3.88	3,426	9.4	Graph of Skills	3.11	3,436	7.0
GoSkills	3.42	3,483	8.2	GoSkills	3.95	2,956	6.0

Table 28: Terminal-Bench 2 sample SDs across three repeats for Table 1. Accuracy is in pp, tokens are cumulative input tokens, and time is agent-only seconds.

Method	Accuracy SD	Token SD	Time SD	Method	Accuracy SD	Token SD	Time SD
GPT-5.4				**Gemini 3 Pro**			
No Skills	4.90	8,873	16.8	No Skills	4.05	15,435	17.7
Vanilla Skills	4.05	62,098	29.9	Vanilla Skills	3.61	52,327	27.8
Vector Skills	5.15	9,030	20.9	Vector Skills	5.19	18,775	19.1
Graph of Skills	1.95	10,057	17.9	Graph of Skills	3.61	19,771	13.6
GoSkills	3.37	10,647	29.1	GoSkills	4.68	17,042	23.3
MiniMax M2.7				**Claude Sonnet 4.5**			
No Skills	3.24	47,169	34.1	No Skills	2.34	33,880	28.4
Vanilla Skills	5.31	114,465	63.4	Vanilla Skills	4.54	93,560	41.6
Vector Skills	3.95	49,598	40.0	Vector Skills	3.95	36,349	34.0
Graph of Skills	2.59	41,891	40.4	Graph of Skills	5.66	32,707	24.4
GoSkills	4.25	51,651	50.9	GoSkills	4.25	32,383	30.8
Qwen 3.5				**Kimi K2.5**			
No Skills	4.68	28,741	60.8	No Skills	3.43	32,436	33.9
Vanilla Skills	4.54	59,104	85.0	Vanilla Skills	5.77	86,070	47.1
Vector Skills	4.90	26,417	61.7	Vector Skills	2.83	41,653	40.6
Graph of Skills	5.31	29,271	57.9	Graph of Skills	2.59	35,784	34.3
GoSkills	3.43	36,581	79.6	GoSkills	3.89	33,169	43.6

G Qualitative Analysis

Section framing. We use trajectory-grounded case studies to explain when group-structured retrieval changes the downstream trajectory and when it does not. The relevant question is not only whether a method retrieves a topically related skill, but whether the exposed context gives the downstream agent an executable entry point, supporting artifacts, and visible checks early enough to change the trajectory. This is the intended difference between Graph of Skills and GoSkills: Graph of Skills scores skill nodes through graph diffusion and hydrates a dependency-aware bundle, whereas GoSkills selects anchor-centered groups, expands support groups, bottlenecks the selected plan to a few atomic payloads, and renders explicit Start, Support, Check, Avoid, and debt fields.

Table 29: Failure taxonomy for group-structured skill retrieval experiments. The taxonomy separates what GOSKILLS can directly affect from downstream and infrastructure bottlenecks.

Error mode	Typical symptom	Interpretation
Activation miss	No activated group contains the skill or facet needed by the visible task requirement.	Retrieval-side failure; better schema extraction, seed evidence, or group indexing can help.
Partial coverage	The anchor is plausible, but the final bottleneck omits a required support, artifact, or visible-check skill.	Retrieval-side failure; this is the purpose of coverage debt and budgeted backfill.
Good retrieval, bad execution	The exposed skills are plausible, but the agent over-builds, ignores the contract, or fails to satisfy the task check.	Mostly downstream; retrieval can reduce search friction but cannot guarantee execution.
Context overhead	The contract or support context adds tokens without improving the trajectory.	Efficiency failure; most likely on long setup chains that need broad environment context.
Infrastructure failure	Container startup, dependency, or evaluator failures prevent a valid task outcome.	Excluded from aggregation. Agent execution timeouts remain valid task failures and contribute their observed token and runtime costs.

Evidence sources. The cases use GPT-5.4 SkillsBench runs from repeat 2 of the final evaluation. For each task we inspect the context each method presented (the retrieval bundle), the agent trajectory, and the verifier output; all numbers in Table 30 are taken from the run records. We chose the cases to cover two wins, a tie, and a loss for GOSKILLS. Among the reward flips we inspected, we excluded those whose trajectories show that the decisive difference was not influenced by the presented context, such as a different reading of an ambiguous requirement or a hand-picked numerical threshold.

Reading guide

Each case is interpreted along three axes: whether the initial context exposes a credible entry point, whether supporting artifacts and checks are present, and whether the remaining failure is better attributed to retrieval, execution, or infrastructure.

Table 30: Matched GPT-5.4 SkillsBench cases from repeat 2. Each method cell reports evaluator reward (percent), agent turns, input tokens, and agent-only time (seconds).

Task	Case type	Graph of Skills	GOSKILLS
`protein-expression-analysis`	Win (mechanism)	0%; 16; 172,681; 102.4s	100%; 9; 58,427; 80.5s
`threejs-structure-parser`	Win (suggestive)	0%; 7; 45,425; 71.3s	100%; 7; 55,709; 85.7s
`3d-scan-calc`	Tie	100%; 8; 38,671; 69.2s	100%; 7; 30,258; 73.3s
`energy-market-pricing`	Loss	100%; 13; 115,680; 185.7s	0%; 12; 99,391; 155.3s

Case 1 · `protein-expression-analysis`: the presented excerpt decides the outcome

Presented. Both methods surface the same spreadsheet skills (`xlsx`, Excel Automation, Excel Analysis). Graph of Skills lists `xlsx` first and presents the head of its skill file: front matter and formatting conventions, with recalculation mentioned only in the front-matter description. GoSkills anchors Excel Automation, lists `xlsx` as support, and presents the `xlsx` section that requires running `recalc.py` after writing formulas.

Trajectory. Both agents write INDEX/MATCH and STDEV formulas with `openpyxl`. The GoSkills agent runs `python3 .../xlsx/recalc.py protein_expression.xlsx 30` at step 7. The Graph of Skills agent never calls `recalc.py`; at step 12 it converts the workbook with LibreOffice into `/tmp`, leaving the original file without cached formula values.

Outcome. GoSkills passes all six tests. Graph of Skills fails the three computation tests ("Only 0/100 cells filled"), using 16 turns and 172,681 input tokens, versus 9 and 58,427 for GoSkills.

Reading. The decisive difference is which part of a shared skill reaches the agent, not the START line. This follows from the source-preserving presentation policy (Appendix D.1), which keeps task-relevant sections within the payload cap.

Case 2 · `threejs-structure-parser`: a surfaced implementation detail

Presented. Both contexts include the `threejs` and `obj-exporter` skills. Only the GoSkills context contains the export rule `if (geom.index) geom = geom.toNonIndexed()`; it also carries two unrelated support skills.

Trajectory. Both agents read the same reference export script. The Graph of Skills agent sees the `toNonIndexed` line while reading it (step 5), but its own export code omits the call. The GoSkills agent writes the call into its export script (step 6).

Outcome. Graph of Skills fails the part- and link-mesh tests (Chamfer distance 1.493 against a threshold of 0.0002); GoSkills passes.

Reading. Suggestive rather than conclusive: we did not re-execute the export to isolate the effect of the indexing step.

Case 3 · `3d-scan-calc`: same entry point, same answer

Presented. Both methods lead with `mesh-analysis`. GoSkills presents two skills in 3,794 characters, and Graph of Skills presents four skills in 8,930 characters; each context includes unrelated extras.

Trajectory. Both agents read the skill and its mesh tool and call the same largest-component analysis.

Outcome. Both return the same mass (34,648.04) and material ID (42). GoSkills uses 7 turns and 30,258 input tokens, versus 8 and 38,671 for Graph of Skills.

Reading. When the baseline already exposes the right entry point, the contract mainly trims context rather than changing the outcome.

Case 4 · `energy-market-pricing`: a loss at a tolerance boundary

Presented. Both contexts contain overlapping power-system skills; GoSkills anchors `power-flow-data`. Its CHECK field lists only artifact facets (`network.json`, `report.json`, and JSON output), and its DEBT field still lists `report.json`.

Trajectory. Neither agent opens a skill file. Both write a DC optimal power flow with reserve coupling using `linprog` with the HiGHS solver.

Outcome. The GoSkills run reports a base cost of 2,965,331.83, below the reference optimum of 2,965,637.91 by more than the 0.01% tolerance (296.56), and fails one optimality test; the Graph of Skills run passes.

Reading. The contract carries no modeling content for this task, and the outcome flips on a small formulation difference at the tolerance boundary.

Scope of the case evidence

These are single-repeat examples chosen to cover two wins, a tie, and a loss. They illustrate how the presented context can change, or fail to change, a trajectory; the paired aggregate results in Appendix E provide the broader comparison across tasks.

H Additional Weaknesses

This section summarizes the main limitations of the current implementation and evaluation protocol. These weaknesses do not change the central goal of GOSKILLS: studying whether group-structured retrieval can make skill contexts more usable under a bounded prompt budget. However, they clarify where the reported results should be interpreted with care.

Dependence on deterministic schema extraction. GOSKILLS relies on deterministic query-schema extraction, typed skill metadata, and visible requirement cues. This design makes the retrieval process transparent and reproducible, but it may be less robust when task prompts are ambiguous, metadata are sparse, or the relevant requirements are only implicit. In particular, the method can miss support skills when a task does not expose clear technology anchors, artifact names, output formats, or failure cues. The current evaluation therefore best reflects settings where task-visible requirements can be extracted reliably before execution.

Fixed scoring rules. The group, support, and bottleneck scores are fixed weighted sums over handcrafted feature families. This makes the method easy to inspect and keeps the retrieval policy unchanged across backbones and benchmarks. However, the coefficients may implicitly fit the development distribution and may not be optimal for other skill libraries or task domains. A learned or validation-tuned scoring policy could improve transfer, but would also introduce additional training data requirements and make it harder to isolate the effect of group-structured retrieval itself.

I Broader impacts

This work studies inference-time retrieval and contextualization for existing agent skill libraries. Potential positive impacts include more efficient use of large skill libraries, bounded agent-facing context, and more transparent agent-facing context through explicit START, SUPPORT, CHECK, AVOID, and DEBT fields. Potential negative impacts include over-reliance on retrieved guidance, more capable automated coding agents being used for unsafe automation, and amplification of unsafe or low-quality skills if such skills are present in the underlying library. The method does not itself filter unsafe skills, verify semantic correctness, or prevent misuse; deployment should therefore pair retrieval with library curation, policy checks, and task-level monitoring.

J Compute resources and API usage

All experiments are API-based agent evaluations. We do not train or fine-tune any model and do not use GPUs. The benchmark workers run the downstream agent loop, retrieval code, logging, and task evaluators on CPU machines. Each worker has 8 vCPUs and 32 GB of RAM, and all experiments run on Ubuntu 22.04 cloud VMs with the same software environment and benchmark versions across methods.

For each model–method–benchmark combination, we run three repeats. The main tables report mean cumulative input tokens and mean agent-only task-processing runtime, excluding environment setup. Infrastructure failures, task timeouts, and valid task outcomes are tracked separately in the run provenance table. API-based model inference is performed through the corresponding provider endpoints; access dates are given in the references.

K Assets, licenses, and terms of use

Table 31 lists the external assets used in this paper with their source, the version used, and their license. All benchmarks, baseline code, and skill libraries are used unmodified and only for research

evaluation, in accordance with their licenses; pinned versions and checksums are recorded in the repository's `DATA.md`. Individual `SKILL.md` files in the downloaded libraries retain the terms of their original sources. Backbone models and the embedding model are accessed through their providers' public APIs under the providers' terms of service, and no model weights are redistributed.

Table 31: External assets used in this paper, with source, pinned version where applicable, and license.

Asset	**Role in this paper**	**Source and version**	**License / terms**
SkillsBench (Li et al., 2026c)	Technical-task benchmark with curated skills and deterministic verifiers.	`benchflow-ai/skillsbench`, commit `7c3d109`	Apache-2.0
ALFWorld (Shridhar et al., 2021) (built on TextWorld)	Multi-turn interactive benchmark.	`alfworld/alfworld`, official downloader; `microsoft/TextWorld`	MIT (ALFWorld and TextWorld)
Terminal-Bench 2 (Merrill et al., 2026)	Terminal and CLI agent benchmark.	`harbor-framework/terminal-bench-2`, commit `69671fb`	Apache-2.0
Harbor	Runs Terminal-Bench 2 task containers and their native verifiers; the agent loop is our own.	`harbor-framework/harbor`, version 0.20.0	Apache-2.0
Graph of Skills (Li et al., 2026a)	Structural retrieval baseline and typed skill-graph constructor.	`davidliuk/graph-of-skills`, commit `203f60a`	MIT
Skill libraries (`skills_200`–`skills_2000`)	Skill corpora for the retrieved-skill methods, and for Vanilla Skills on ALFWorld and Terminal-Bench 2.	Hugging Face `DLPenn/graph-of-skills-data`, revision `83a8c59`	MIT
Agent Skills (Anthropic, 2026)	Skill package format; Vanilla Skills baseline.	`agentskills/agentskills`	Apache-2.0 (code), CC-BY-4.0 (docs)
`text-embedding-3-large` (OpenAI, 2024)	Embedding model for Vector Skills.	OpenAI API	OpenAI terms of use
Backbone models	GPT-5.4, Claude Sonnet 4.5, Gemini 3 Pro, MiniMax M2.7, Qwen 3.5, Kimi K2.5.	Provider APIs	Provider terms of service
GoSkills code (ours)	Group construction, retrieval, contract rendering, and evaluation scripts.	`yuhuo03/Group-of-Skills`	MIT
GoSkills data (ours)	Skill libraries, prebuilt workspaces, and group pools at 200–2,000 skills.	Hugging Face `yuhuo03/group-of-skills-data`	Upstream skill licenses

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state that GOSKILLS is an inference-time contextualization layer and limit the claims to the evaluated benchmarks and agent settings. We report both effectiveness and efficiency results in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the main assumptions and limitations in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present formal theoretical results or proofs; the proposed scoring functions and algorithms are used as empirical design components.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 describes the benchmarks, baselines, evaluation metrics, model backbones, skill budgets, and run protocol used to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The public repository linked below the author list provides retrieval code, baseline runners, configuration examples, and evaluation instructions. Appendix E.1 specifies the reported task sets and baseline conditions. The release's default task manifest and skill-directory configuration differ from some reported-run settings and require alignment for reproduction.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We specify the evaluation benchmarks, baselines, inference settings, retrieval budgets, scoring components, and metrics in Section 4 and provide additional details in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Appendix E reports task-level paired bootstrap confidence intervals for the main GPT-5.4 SkillsBench comparisons. We resample paired tasks with replacement for 10,000 bootstrap samples and report percentile 95% confidence intervals for reward and runtime deltas, and the robustness study reports standard deviations over corruption seeds. Appendix E.17 reports sample standard deviations across three repeats for every configuration in the main results and control/ablation tables, separately from task-resampling and corruption-seed variability.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix J describes the compute resources used for the experiments. Our experiments are API-based agent evaluations and do not train models or use GPUs. We report the CPU-based execution workers, memory, execution environment, number of repeated runs, API model backbones, token usage, and agent-only runtime. The main tables report mean cumulative input tokens and agent-only task-processing time, while the run logs track task-level execution status and runtime.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and believe the research conforms to it. The work evaluates an inference-time skill-retrieval interface on benchmark tasks, does not involve human subjects or private data, and does not modify or deploy a system for consequential decisions; its limitations and scope are explicitly stated.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss broader impacts in Appendix I. Potential positive impacts include more efficient use of large skill libraries, bounded agent-facing context, and more transparent agent-facing retrieval through explicit role-labeled contracts. Potential negative impacts include over-reliance on retrieved guidance, amplification of unsafe or low-quality skills already present in the library, and misuse in automated coding-agent workflows. We therefore recommend pairing the method with library curation, policy checks, and task-level monitoring.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk pretrained models, scraped datasets, or generative systems requiring controlled access safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All benchmarks, harnesses, baseline code, skill libraries, and model APIs are cited, and Appendix K (Table 31) lists each with its source, version, and license; pinned versions and checksums are in the repository's DATA.md.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The GoSkills repository provides installation and evaluation instructions in its README. The GoSkills code is released under the MIT License, and the skill libraries, workspaces, and group pools are released as the group-of-skills-data dataset (licenses in Table 31).

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research and therefore does not require IRB approval.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM-based coding agents are part of the evaluated system, and the PH-LLM control uses an LLM-based role annotator. We describe the model backbones and inference protocol in Section 4, the controls in Appendix E.5, and the agent-facing prompts in Appendix E.2.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.