SkillDAG: Self-Evolving Typed Skill Graphs for LLM Skill Selection at Scale

Anonymous ACL submission

Abstract

As LLM agents adopt large skill libraries, selecting the right subset becomes a structural problem rather than a similarity-matching one: skills depend on, conflict with, specialize, or duplicate one another, a structure invisible to both full enumeration and embedding similarity. We present SkillDAG, which models inter-skill relationships as a typed directed graph and exposes it to an LLM agent as an inference-time, agent-callable structural retrieval interface, queried and evolved during execution rather than baked into a fixed retrieval pipeline: each `search` returns vector matches, typed-edge neighbors, and conflict signals, and a propose-then-commit protocol lets the agent register execution-backed edges so the graph accumulates structure across episodes. On ALFWorld and SkillsBench with MiniMax-M2.7, SkillDAG reaches 67.1% success and 27.3% reward, exceeding the strongest reported Graph-of-Skills baseline by +12.8 and +8.6 points; the advantage ports to `gpt-5.2-codex`, and intrinsic SkillsBench Ret@K rises from 65.5 to 78.2 under matched queries. These gains trace to isolable mechanisms: candidate ranking that stays robust as the pool grows 10× where a fixed seeding–diffusion pipeline degrades, and set-monotone online edits that enlarge ground-truth recall without evicting prior hits.

1 Introduction

Large language model agents increasingly rely on external skill libraries to extend their capabilities beyond the procedural knowledge in model weights. A skill is a self-contained package: a unique identifier, a natural-language description of when it applies, and a body of instructions or code the agent runs when invoked, often with auxiliary scripts and prerequisite notes (Liu et al., 2026). Voyager (Wang et al., 2023a), JARVIS-1 (Wang et al., 2023b), and ToolLLM (Qin et al., 2024) report large gains once an agent is equipped with a curated skill collection. As collections grow to thousands of entries, the bottleneck shifts from authoring to skill *selection*: which subset to load cannot be decided from a name or embedding alone, and retrieval over large tool collections is itself a measurable failure mode of LLM agents (Shi et al., 2025).

Two strategies dominate practice. Concatenating the whole library is feasible only at small scale: token cost grows linearly and descriptions are lost in an overloaded prompt (Liu et al., 2024a). Flat retrieval instead embeds the task and concatenates the top-K skills by cosine similarity (Karpukhin et al., 2020; Reimers and Gurevych, 2019) or BM25 (Robertson and Zaragoza, 2009). Both hide the library's relational structure: the top match names the task while a functionally necessary prerequisite (a parser, converter, or setup utility) is silently omitted; analogous gaps arise from skills that interfere when co-selected, near-duplicates that waste context budget, and pairs that jointly enable a capability neither provides alone. Curation can mask these failures at small scale, but its cost grows with the library.

Prior work models inter-skill structure but stops short of letting the LLM reason over it directly. SkillNet (Liang et al., 2026) builds a 500,000-skill graph used for platform organization rather than runtime routing. Graph of Skills (GoS; Liu et al., 2026) runs Personalized PageRank (Page et al., 1999; Haveliwala, 2003) over a typed skill graph at inference time and reports gains on SkillsBench and ALFWorld, but computes the bundle through a fixed seeding–diffusion–reranking pipeline and hands the agent opaque context: it cannot ask why

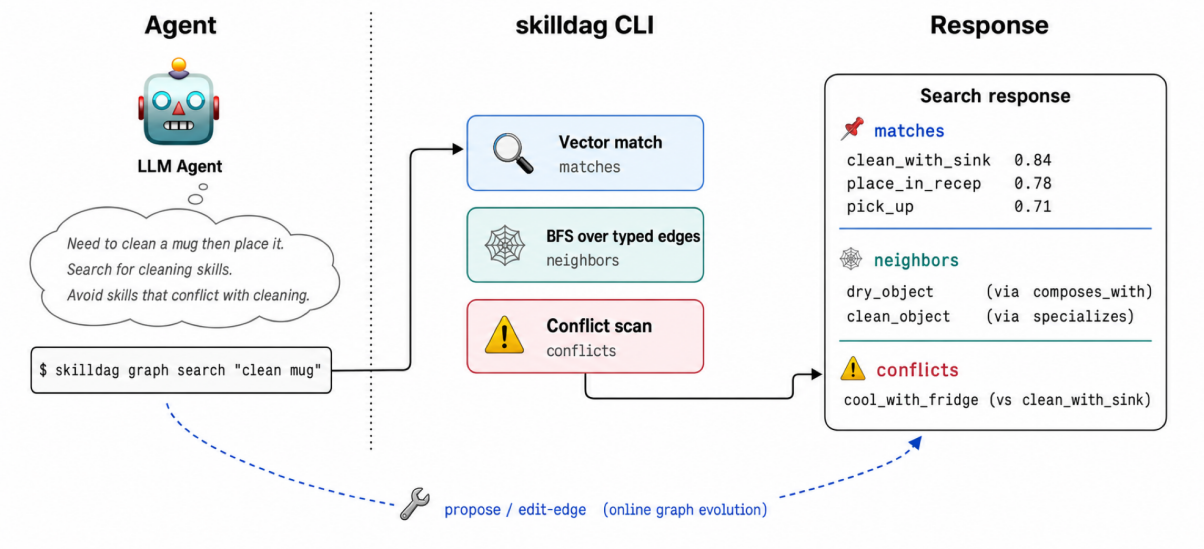


Figure 1: SkillDAG per-episode workflow. The agent issues `search` to obtain a three-field response (matches, neighbors, conflicts), calls `show` to fetch any skill body, acts in the environment, and may commit an evidence-backed graph edit through the `propose-edge`/`edit-edge` pair. Every commit passes acyclicity and non-contradiction checks with an append-only log enabling rollback; the updated graph feeds subsequent `search` calls. The LLM, not a graph-ranking policy, decides what to load.

a skill was chosen, retract a noisy retrieval, or record a relationship the offline pipeline missed. XSkill (Jiang et al., 2026) and CUA-Skill (Chen et al., 2026) address skill acquisition and execution respectively, not inter-skill relations.

We propose SkillDAG, built on the inverse design choice: rather than the substrate of a fixed retrieval algorithm, the graph is exposed to the LLM as an inference-time, agent-callable structural retrieval interface, on the same footing as the tools the agent already uses for execution. The LLM, not a graph-ranking policy, decides how structural evidence should affect execution; the graph in turn accumulates structural knowledge across episodes through evidence-backed agent edits.

Our contributions are summarized as follows:

- We identify that flat retrieval over large skill libraries silently omits structurally necessary skills and concatenates redundant or interfering ones—a failure mode invisible to similarity matching alone.
- We propose SkillDAG, an agent-callable structural retrieval interface that exposes a typed skill relation graph to the LLM via `search`/`show`/`propose-edge`/`edit-edge`, replacing a fixed graph-ranking policy with structural evidence the agent reasons over directly.
- We design an online propose-then-commit edit protocol under three structural invariants (acyclicity, non-contradiction, append-only reversibility) that grows the graph from single-episode evidence, paired with a two-view cold-start constructor that recovers cross-functional pairs flat self-similarity misses.
- Experiments on ALFWorld (Shridhar et al., 2021) and SkillsBench (Liu et al., 2026) with MiniMax-M2.7 and `gpt-5.2-codex` show SkillDAG reaches 67.1%/93.6% ALFWorld success and 27.3%/36.8% SkillsBench reward, with intrinsic Ret@K of 78.2 stable across a 10× skill-pool expansion.

2 Related Work

Skill and tool libraries. LLM agents are augmented with external capabilities exposed as tools (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024), skills (Wang et al., 2023a,b), code actions (Liang et al., 2023; Wang et al., 2024b), or grounded primitives (Ahn et al., 2022; Huang et al., 2022). As libraries scale, selection rather than authoring becomes the bottleneck: ToolkenGPT (Hao et al., 2023) learns per-tool embeddings, AnyTool (Du et al., 2024) adds a hierarchical API

retriever, CRAFT (Yuan et al., 2024) creates and retrieves specialized tool-sets, and library-learning methods compress trajectories into documented reusable code (Grand et al., 2024). SkillNet (Liang et al., 2026) organizes 500K+ skills as a graph for developer navigation, CUA-Skill (Chen et al., 2026) models skills as executable DAGs, and GraSP (Xia et al., 2026) adds DAG repair for intra-task sequencing; none expose inter-skill structure to the agent at runtime, so flat selection still misses prerequisites and conflicts as the library grows.

Skill retrieval and structured RAG. Beyond flat top-K RAG (Lewis et al., 2020), recent work injects structure: GraphRAG (Edge et al., 2024) indexes a community graph for offline summarization, HuggingGPT (Shen et al., 2023) composes models by task decomposition over a flat catalog, HippoRAG (Gutiérrez et al., 2024) retrieves by Personalized PageRank over a KG memory, LightRAG (Guo et al., 2025) maintains an incrementally updated graph index, and GeAR (Shen et al., 2025) performs agent-driven graph expansion. For tools specifically, Re-Invoke (Chen et al., 2024) rewrites queries for zero-shot retrieval and ToolGen (Wang et al., 2025) folds lookup into generation via tool tokens. Graph of Skills (GoS; Liu et al., 2026), the closest baseline, builds a typed skill graph offline and retrieves a bundle by reverse-aware Personalized PageRank that is concatenated into context as opaque text. SkillDAG inverts two choices: the graph is an agent-callable structural retrieval interface rather than a fixed retriever's substrate, and its edges are editable during execution rather than frozen at construction.

Self-evolving agent. ReAct (Yao et al., 2023b) interleaves reasoning (Wei et al., 2022) and action, extended by deliberate search (Yao et al., 2023a); Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) add verbal self-correction; ExpeL (Zhao et al., 2024), AutoAct (Qiao et al., 2024), and Agent-Pro (Zhang et al., 2024) distill experience into reusable insights or policies. Closer to skill accumulation, Agent Workflow Memory (Wang et al., 2024c) induces reusable workflows online, ICAL (Sarch et al., 2024) and STE (Wang et al., 2024a) abstract trajectories into memory, AutoGuide (Fu et al., 2024) retrieves state-conditioned guidelines, and OS-Copilot (Wu et al., 2024) accumulates skills in a tool-rich OS; surveys frame these as non-parametric self-evolution (Gao et al., 2025). Generative Agents (Park et al., 2023) and MemGPT (Packer et al., 2023) manage free-text memory, and XSkill (Jiang et al., 2026) updates individual skills. Our propose/edit-edge mechanism instead mutates a typed relational structure shared by all future agents, enabling precise, auditable updates.

Agent benchmarks. We evaluate on ALFWorld (Shridhar et al., 2021) and SkillsBench (Liu et al., 2026), within a landscape spanning cross-domain tasks (Liu et al., 2024b), web environments (Zhou et al., 2024), general assistants (Mialon et al., 2024), desktop/OS control (Xie et al., 2024), API-rich apps (Trivedi et al., 2024), tool-agent-user dialogue (Yao et al., 2024), and function calling (Patil et al., 2025). All share the pressure point motivating SkillDAG: retrieval over a growing capability surface is itself a measurable failure mode (Shi et al., 2025).

3 Method

3.1 Overview

SkillDAG starts from a simple inversion: the typed graph is not hidden inside a fixed retrieval pipeline, but exposed as an agent-callable interface the LLM can query, inspect, and revise during execution. This makes graph structure an explicit part of the agent's decision context rather than an opaque ranking heuristic. The typed relation graph gives otherwise flat skill libraries operational structure: prerequisite and specialization edges organize reusable progressions, composition and similarity edges surface adjacent skills, and conflict edges mark combinations the agent should avoid. The `search`/`show` interface turns that structure into action by returning semantic matches, typed neighbors, and conflicts as separate evidence channels while loading full skill bodies only on demand. To initialize such structure before any episode runs, the cold-start constructor links skills from two complementary views: what a skill does and what it needs. Static structure is inevitably incomplete, however, so the online protocol lets the agent propose and commit execution-backed edits while acyclicity,

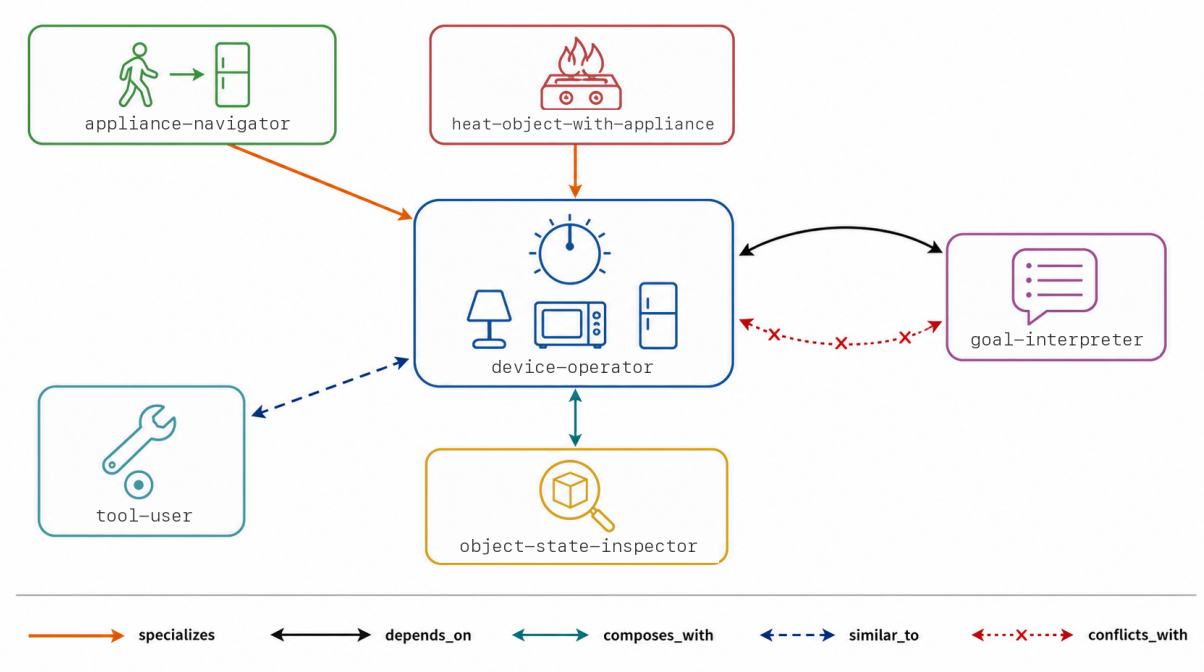


Figure 2: A six-node subgraph from the post-run ALFWorld graph exercising all five typed relations: cold-start `specializes`/`similar_to` edges plus an agent-registered `depends_on` and a later `conflicts_with` on the same pair.

non-contradiction, and append-only reversibility keep the graph from drifting into incoherence. The formal graph, interface, cold-start constructor, and online edit protocol are specified in §3.2–§3.5, with full pseudocode in Appendix A.

3.2 Skill Relation Graph Definition

Prerequisites, alternatives, and conflicts collapse onto a single cosine score under the default retrieval baseline (rank skills by self-description similarity and concatenate the top-K), so a hard prerequisite and an interchangeable alternative both surface as "high-cosine neighbors," even though loading the former unlocks the task while loading the latter wastes context and dilutes the agent's attention. To prevent this collapse, we elevate the relation *type* itself to be part of the agent-facing semantics, so the LLM acts on *what kind of* relation it sees rather than merely *how similar* two skills look. We instantiate this principle as a typed graph. Let the skill library be a finite set $\mathcal{C} = \{s_1, \ldots, s_n\}$ of skill packages, where s_i is also the unique identifier; the graph $G = (V, E, \tau)$ has $V = \mathcal{C}$ and a typing function τ assigning each edge one of five operationally specified types: `depends_on`(A, B) marks A as requiring the prerequisite B; `specializes`(D, A) marks D as the narrower variant to prefer over A; `composes_with` marks synergistic co-use; `similar_to` marks functional redundancy (pick one); and `conflicts_with` marks pairs that should not be co-selected.

Two operational constraints make this typed graph usable as an agent-facing interface. *Conflicts as a pruning signal, not a navigation edge.* We deliberately exclude `conflicts_with` from traversal because navigating a conflict edge would actively undermine retrieval by surfacing the very skills the LLM has been told to avoid; the four positive types are walked by the retrieval interface, while `conflicts_with` is only used to exclude co-selection (§3.3). *Acyclicity on the backbone.* We reserve `depends_on` and `specializes` as the directed backbone whose acyclicity gives SkillDAG its name; the invariant is enforced at commit time (§3.5). Figure 2 shows a six-node post-run ALFWorld subgraph exercising all five types.

3.3 Agent-Callable Structural Retrieval Interface

Having typed the edges, we must decide how to expose the graph to the agent at inference time. The default move in prior graph retrievers

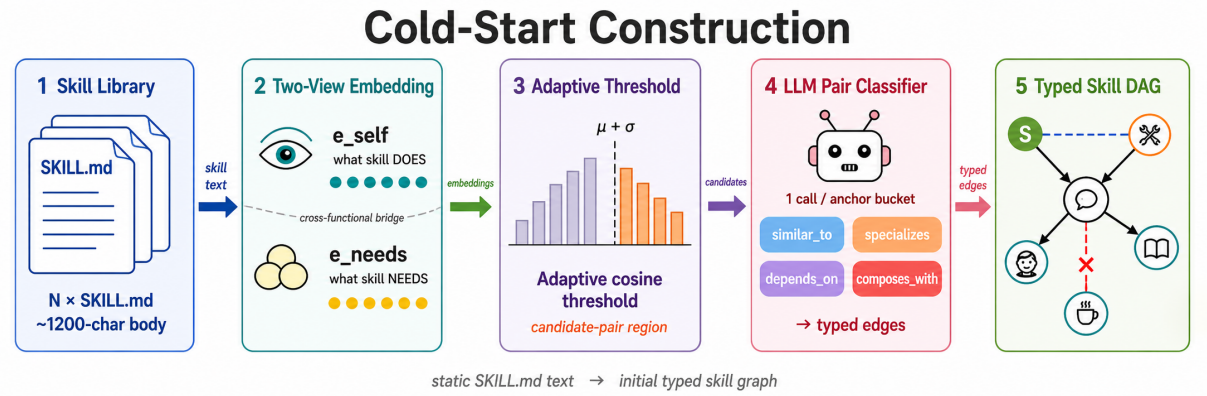


Figure 3: Cold-start construction: read the library, embed each skill in two views (e_{self} for what it does, e_{needs} for what it requires), apply an adaptive cosine threshold, run an LLM pair classifier, and materialize the initial graph. `conflicts_with` is reserved for the online protocol because co-use harm cannot be inferred from static text alone.

is to collapse typed structure back into one ranked bundle, but doing so would undo the work of §3.2, hiding from the LLM exactly the distinctions we built the graph to preserve. We therefore designed `search` (Figure 1) as an agent-callable interface that returns semantic matches, typed neighbors, and conflicts as three separate evidence channels the LLM reasons over independently.

A single `search`(q, K, D) call returns three fields, each answering a distinct question the LLM would otherwise have to ask separately: **matches** ("what is topically close to q?"), the top-K skills by query–node embedding cosine (Karpukhin et al., 2020); **neighbors** ("what does the typed graph attach to those matches?"), skills reached by bounded breadth-first traversal of depth D over the four walkable edge types; and **conflicts** ("what should I not co-load with them?"), the one-hop `conflicts_with` edges incident to any match.

We chose to keep these channels separate, rather than fusing them into a single score, because the decomposition gives the agent three capabilities that a ranked bundle cannot offer: it can drop an entire field under a tight context budget (e.g., skip **neighbors** when context is scarce), schedule `search` calls on its own initiative (e.g., re-query after partial progress), and leave a reasoning trace documenting why each skill was selected or excluded.

3.4 Cold-Start Graph Construction

The interface of §3.3 assumes a graph already exists; we now describe how it is first populated, before the agent ever runs. The obvious choice (embed each skill's self-description and link nearest neighbors) has a specific failure mode we set out to fix: skills that should be connected often describe different actions, where one provides what another requires. A cooling skill and a pickup skill, for example, have disjoint self-descriptions, yet the cooling skill requires "object in hand" produced by the pickup skill. Self-description similarity alone systematically misses such cross-functional pairs, so we designed a two-view embedding strategy that initializes the graph from static `SKILL.md` text while explicitly recovering these bridges.

The key departure from single-view retrieval is the second embedding. For each skill v, $e_{\text{self}}(v)$ captures *what a skill does* (identifier, description, body preview), while $e_{\text{needs}}(v)$ captures *what it requires*: an LLM imagines a few invoking tasks and summarizes the shared prerequisites in one sentence (HyDE-style (Gao et al., 2023)), then embeds the summary. e_{self} surfaces topical neighbors; e_{needs} surfaces functional bridges, and a candidate pair appears if either ranking exceeds the adaptive cosine threshold. An LLM classifier (Zheng et al., 2023) then assigns one of the four static types or `none`. Figure 3 traces these stages end to end: read library, embed in two views, threshold, classify, materialize.

We reserve `conflicts_with` for the online protocol (§3.5) by operational necessity: a conflict is, by definition, a pair whose co-use predictably degrades success, and that evidence simply does not exist until execution is ob-

served. No amount of static reading recovers it.

3.5 Online Graph Evolution

Even a well-constructed cold-start graph is necessarily incomplete. By the argument just made it cannot contain `conflicts_with` edges at all; more broadly, many useful relations appear only during execution: a prerequisite the cold-start LLM did not anticipate, an interference pair observable only when both skills are loaded together, a refinement of an existing relation after a near-miss. To close this gap, we make graph construction an online part of agency rather than offline preprocessing: the per-episode loop in Figure 1 includes a `propose-edge`/`edit-edge` pair through which the agent commits evidence-backed edits at the same pace it acts in the environment.

We split the edit into two calls so that committing is always a deliberate act rather than a side effect of execution. `propose-edge` performs a dry run: it previews the change and surfaces existing edges and history for the same skill pair, so the agent can reason about whether to commit. `edit-edge` then commits the change with a natural-language `reason` and the supporting execution trace; the same call can add, delete, or retype an existing edge, giving the agent an explicit retraction path for stale or mistaken relations. We deliberately chose to gate commits only by the agent's own reasoning, not by a downstream voting threshold or a human curator, so that a single execution episode suffices and the graph evolves at agent speed rather than curation speed. The repeated pair in Figure 2 illustrates why the preview surfaces pair history: a later `conflicts_with` decision must be evaluated against earlier `depends_on` evidence on the same pair.

Gating commits only by the agent's reasoning raises the risk of single-shot corruption, which we bound with three structural invariants every commit must pass; the invariants are chosen to limit the blast radius of a bad edit without second-guessing the agent's semantic judgment. *Acyclicity*: a `depends_on`/`specializes` commit that would close a backbone cycle is rejected, since a circular prerequisite chain would loop the agent through skills that each demand the next, voiding the DAG property declared in §3.2. *Non-contradiction*: a pair cannot simultaneously carry `conflicts_with` and a positive surfacing edge, preventing the navigation and pruning signals from cancelling each other. *Reversibility*: an append-only history log supports rollback by recency, so a spurious edit is bounded and recoverable. These invariants are structural rather than semantic by design: they preserve the LLM's authority over what each edge *means* while preventing the graph from drifting into incoherence.

4 Experiments

Our evaluation pursues four guiding questions about when and why the interface design pays off. We first compare SkillDAG against full-library prompting, vector retrieval, and GoS on end-task reward—the strongest indicator that agent-callable graph access actually beats the dominant baselines. Replaying identical queries through both retrievers then isolates how much of the gain is already visible before any downstream execution. Contrasting MiniMax-M2.7 with `gpt-5.2-codex` shows how the effect changes with backbone reasoning strength. Finally, a 10× skill-pool expansion and an ALFWorld train/test split measure whether retrieval stays robust at scale and whether online edits transfer to held-out episodes. Setup is in §4.1, main results in §4.2, and ablations in §4.3.

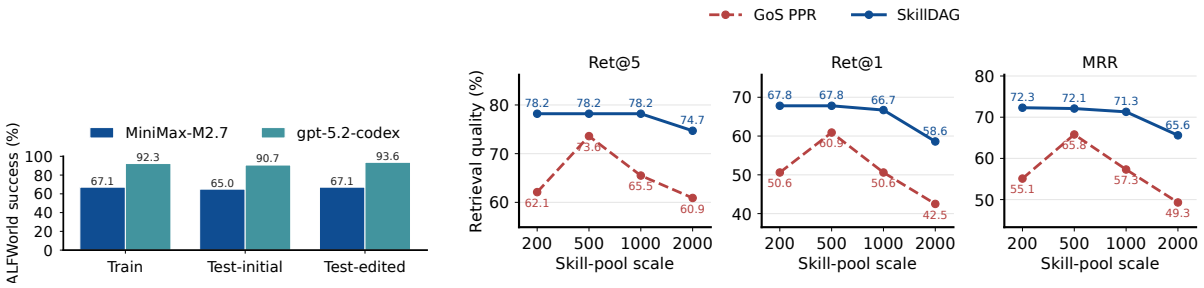
4.1 Experimental Setup

Benchmarks. We evaluate on two benchmarks used by Liu et al. (2026), which exercise different aspects of skill selection at scale. *ALFWorld* (Shridhar et al., 2021) is a text-based embodied environment whose six task types (pick-and-place, examine-in-light, clean-, heat-, cool-and-place, pick-two-and-place) require composing navigation, manipulation, and state-change behaviors; we run the 140-episode `valid_seen` split with a 37-skill procedural library. *SkillsBench* (Liu et al., 2026) is 87 containerized code-generation tasks scored by unit-test pass fraction, shipped at four skill-pool scales (200/500/1000/2000 `SKILL.md` documents) so retrieval quality can be studied as the pool grows.

Baselines. We compare against three baselines from Liu et al. (2026). *Vanilla Skills* places the

Table 1: Main results. `R` = reward (%); ALFWorld `R` = success rate. SkillsBench `R` at 1000-skill scale. `Ret@K`/`Ret@1`/MRR (↑) are intrinsic retrieval metrics. Vanilla/Vector/GoS `R` quoted from Liu et al. (2026) Table 1; SkillDAG `R` from our runs. Best per column in bold.

		SkillsBench				ALFWorld			
Model	**Method**	R↑	Ret@K↑	Ret@1↑	MRR↑	R↑	Ret@K↑	Ret@1↑	MRR↑
MiniMax-M2.7	Vanilla Skills	17.2	–	–	–	47.1	–	–	–
	Vector Skills	10.4	10.8	3.6	5.8	50.7	68.6	37.9	49.2
	GoS	18.7	65.5	50.6	57.3	54.3	86.4	56.4	67.9
	SkillDAG	27.3	**78.2**	66.7	71.3	67.1	**92.1**	**57.9**	**71.1**
gpt-5.2-codex	Vanilla Skills	27.4	–	–	–	89.3	–	–	–
	Vector Skills	21.5	10.8	3.6	5.8	92.9	68.6	37.9	49.2
	GoS	34.4	65.5	50.6	57.3	**93.6**	**86.4**	56.4	67.9
	SkillDAG	**36.8**	75.9	**70.1**	**73.0**	**93.6**	85.1	**60.4**	**69.6**



(a) ALFWorld train/test.

(b) Cross-scale retrieval on SkillsBench.

Figure 4: Robustness experiments. (a) ALFWorld success (%) across Train (420 in-domain, edits on), Test-initial (140 held-out on cold-start graph), and Test-edited (same 140 on train-produced graph), for both backbones. (b) Same 87 SkillDAG queries through both retrievers, shared `text-embedding-3-large`; SkillDAG outranks GoS at every scale and stays flat across the 10× expansion; scale-1000 matches the MiniMax row of Table 1.

entire library in the prompt and acts via ReAct (Yao et al., 2023b) (the recall upper bound and context-cost reference). *Vector Skills* concatenates the top-K embedding-cosine matches, using no graph structure. *Graph of Skills* (GoS) (Liu et al., 2026) retrieves a bundle by hybrid semantic–lexical seeding plus reverse-aware Personalized PageRank over a typed graph, reranked under a token budget, then concatenated into the prompt. *SkillDAG* (ours) exposes the graph as an agent-callable structural retrieval interface: the agent issues `search`/`show`/`propose-edge`/`edit-edge` during execution, acts on the three-field search response, and commits evidence-backed edges under the guardrail checks.

Models & evaluation. We report per-episode reward `R` (%); for ALFWorld `R` equals success rate. The `Vanilla`/`Vector`/GoS `R` rows (for both MiniMax-M2.7 and `gpt-5.2-codex`) are quoted from Liu et al. (2026) Table 1 and were not rerun locally; the SkillDAG `R` rows are our own runs on the same benchmark, backbone, task data, and skill pools, but are not paired seed-matched reruns of the baselines. The intrinsic retrieval columns are computed by us on identical inputs: the actual `search` queries issued by the SkillDAG agent during its 87-task run, evaluated by each pipeline on the scale-1000 pool (n=140 ALFWorld episodes, n=87 SkillsBench tasks).

4.2 Main Results

Table 1 reports task reward and intrinsic retrieval on both benchmarks and backbones.

SkillDAG improves most under MiniMax-M2.7: ALFWorld 54.3 → 67.1 (+12.8) and SkillsBench 18.7 → 27.3 (+8.6), while under `gpt-5.2-codex` ALFWorld ties at 93.6% and SkillsBench narrows to 34.4→36.8 (+2.4); the routing aid signature is large where the backbone cannot reason past a poor bundle and vanishes once it can. Intrinsic SkillsBench retrieval likewise improves over GoS on the same MiniMax query distribution: Ret@K 65.5→78.2,

Ret@1 50.6 → 66.7, MRR 57.3 → 71.3, isolating a retrieval-pipeline gain before any agent-side execution; with embedding and queries held fixed, SkillDAG's `matches`-field cosine over typed-graph node embeddings ranks more accurately than GoS's HNSW (Malkov and Yashunin, 2020) seed+PPR (Page et al., 1999) diffusion.

On `gpt-5.2-codex`, SkillDAG ties GoS on ALFWorld (93.6%), remains best on SkillsBench (36.8%), and yields 75.9/70.1/73.0 Ret@5/Ret@1/MRR on codex trajectory queries.

4.3 Scale and Edit-Transfer Ablations

We isolate two effects behind the main results: whether the agent's online edits persist as durable, recall-safe structure, and whether the typed-graph retriever stays robust as the skill pool grows.

We measure online editing two ways: an ALFWorld train/test split that tests whether committed edges transfer, and a SkillsBench cold-vs-edited query replay that tests what an accepted edge does to retrieved recall (Table 2).

Table 2: SkillsBench scale-1000 retrieval, replaying 316 `search` queries from 65 tasks against the cold-start vs. edited graph. "Mean GT/q" is the average number of ground-truth skills retrieved per query (mean ground-truth size 2.52 skills/task).

Graph	Edges	Mean GT/q
Cold-start	2,359	1.915
Edited (+27 online)	2,386	**1.984**

Committed edges persist as durable structure. A 420-episode in-domain *train* run accumulates edits; *test-initial* (140 held-out episodes, cold-start graph) and *test-edited* (same 140, train-produced graph) are then run with edits disabled. On both backbones train and test-initial sit within 2.1 points (MiniMax-M2.7 67.1/65.0/67.1%; `gpt-5.2-codex` 92.3/90.7/93.6%; Figure 4a): transfer holds to the held-out split, and test-edited recovers the train number, so committed edges persist as durable structure rather than training-time stochastics.

Online edits expand ground-truth recall set-monotonically. Replaying all 316 scale-1000 `search` queries, the edited graph (+27 single-observation commits) retrieves more ground-truth skills per query (1.984 vs. 1.915; Table 2) and *zero* queries lose any ground-truth retrieval. This follows by construction: the scorer takes the untruncated union of `matches` and BFS neighbors, so an accepted edge can only enlarge the retrieved set, never evicting a ground-truth skill the cold-start graph already surfaced; this is a recall property, not a context-budget claim.

SkillDAG ranking stays robust as the pool grows 10×**.** With the embedding backbone held fixed, SkillDAG's plain `matches` cosine outranks GoS on every retrieval metric at every pool scale despite using no diffusion step: +16.1/+17.2/+17.2 (Ret@5/Ret@1/MRR) at 200 skills and +13.8/+16.1/+16.3 at 2000, narrowest at 500 (+4.6/+6.9/+6.3) where GoS's PPR pull happens to align with the gold target; across the 10× pool expansion it loses only 3.5 Ret@5 points (78.2 → 74.7), scaling robustly to larger libraries (Figure 4b). This isolates the offline retrieval pipeline, not the typed-edge `neighbors`/`conflicts` fields the agent exercises during execution.

5 Conclusion

SkillDAG exposes a typed skill-relation graph as an agent-callable structural retrieval interface (`search`/`show`/`propose-edge`/`edit-edge`) rather than hiding it inside a fixed graph-ranking policy. The agent loads only the typed neighbors and skill bodies it needs, avoiding the context-bloat failure where fixed-bundle retrievers flood the prompt with marginally relevant skills and corrupt downstream reasoning. Against GoS, SkillDAG lifts MiniMax-M2.7 reward on ALFWorld (54.3 → 67.1) and SkillsBench (18.7 → 27.3), with `matches`-field retrieval winning every metric at every pool scale—supporting the inversion claim that the graph works best as an interface for routing decisions, not as an opaque policy.

Limitations

The propose-then-commit protocol is intentionally agent-driven: a single execution observation can suffice for the agent to commit an edge, bounded by structural guardrails (acyclicity, non-contradiction, reversibility) rather than a statistical confidence threshold (§3.5). Aggregate edit effects are reported (§4.3), but the long-horizon behaviour of single-observation self-evolution remains an open question we view as a direction for future work.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, and Chelsea Finn. 2022. Do as i can, not as i say: Grounding language in robotic affordances. In *CoRL*.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, and Leon Xu. 2026. CUA-Skill: Develop skills for computer using agent. *arXiv preprint arXiv:2601.21123*.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-Invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics: EMNLP*.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. AnyTool: Self-reflective, hierarchical agents for large-scale API calls. In *International Conference on Machine Learning (ICML)*.

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

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, and 8 others. 2025. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *ACL*.

Gabriel Grand, Lionel Wong, Maddy Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. 2024. LILO: Learning interpretable libraries by compressing and documenting code. In *International Conference on Learning Representations (ICLR)*.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Taher H. Haveliwala. 2003. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner monologue: Embodied reasoning through planning with language models. In *CoRL*.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. 2026. XSkill: Continual learning from experience and skills in multimodal agents. *arXiv preprint arXiv:2603.12056*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *ICRA*.

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

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

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, and Kejuan Yang. 2024b. AgentBench: Evaluating LLMs as agents. In *ICLR*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, and Yiming Yang. 2023. Self-Refine: Iterative refinement with self-feedback. In *NeurIPS*.

Yu A. Malkov and Dmitry A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.

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

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *UIST*.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang, Chengfei Lv, and Huajun Chen. 2024. AutoAct: Automatic agent learning from scratch for QA via self-planning. In *ACL*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, and Bill Qian. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world apis. In *ICLR*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *EMNLP*.

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

Gabriel Sarch, Lawrence Jang, Michael J. Tarr, William W. Cohen, Kenneth Marino, and Katerina Fragkiadaki. 2024. VLM agents generate their own memories: Distilling experience into embodied programs of thought. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face. In *NeurIPS*.

Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, Zhongyang Li, Ye Qi, Yang Ren, Dandan Tu, and Jeff Z. Pan. 2025. GeAR: Graph-enhanced agent for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL*.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin,

and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. *arXiv preprint arXiv:2503.01763*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024a. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

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

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025. ToolGen: Unified tool retrieval and calling via generation. In *International Conference on Learning Representations (ICLR)*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better LLM agents. In *ICML*.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024c. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. JARVIS-1: Open-world multi-task agents with memory-augmented multimodal language models. In *arXiv preprint arXiv:2311.05997*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. OS-Copilot: Towards generalist computer agents with self-improvement. In *International Conference on Learning Representations (ICLR)*.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *ICLR*.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. 2024. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations (ICLR)*.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024. AgentPro: Learning to evolve via policy-level reflection and optimization. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM agents are experiential learners. In *AAAI*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *NeurIPS Datasets and Benchmarks*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A realistic web environment for building autonomous agents. In *ICLR*.

A End-to-End Algorithms

Algorithms 1 and 2 together specify the SkillDAG workflow end to end. Algorithm 1 builds the initial typed graph offline from the skill library (§3.4); Algorithm 2 runs the per-episode online loop covering retrieval and graph edits (§3.3, §3.5). The pair classifier ψ returns one of `depends_on`, `specializes`, `composes_with`, `similar_to`, or `none`; `conflicts_with` is reserved for the online phase. Agent-facing calls (`search`, `show`, `propose-edge`, `edit-edge`) and the three commit-time invariants (acyclicity, non-contradiction, and append-only reversibility) appear inline in Algorithm 2.

Algorithm 1 Cold-start graph construction.

Require: Skill library $\mathcal{C}$; embedder ϕ; pair classifier ψ; per-ranking adaptive thresholds $\theta_{\text{self}}, \theta_{\text{needs}}$
Ensure: Initial typed graph $G_0 = (V, E, \tau)$

```
for each v ∈ C do
    e_self(v) ← φ(SKILL.md(v))
    e_needs(v) ← φ(HyDE-summary(v)) {⊥ if self-contained}
end for
P_self ← {(i, j) : cos(e_self(i), e_self(j)) ≥ θ_self}
P_needs ← {(i, j) : e_needs(i) ≠ ⊥ ∧ cos(e_needs(i), e_self(j)) ≥ θ_needs}
E_0 ← ∅;  τ_0 ← empty map
for each (a, b) ∈ P_self ∪ P_needs do
    t ← ψ(a, b)
    if t ≠ none then
        E_0 ← E_0 ∪ {(a, b)};  τ_0(a, b) ← t
    end if
end for
G_0 ← (C, E_0, τ_0) {conflicts_with reserved for online phase}
return G_0
```

Algorithm 2 Per-episode online loop.

Require: Initial graph G_0; task distribution $\mathcal{T}$; LLM agent π; budgets K, D
Ensure: Evolved graph G and append-only history log H

```
G ← G_0;  H ← ∅
for each task T ∈ 𝒯 do
    while episode not terminated do
        π calls search/show as needed, then acts and observes outcome
        if evidence supports an edit Δ then
            (preview, history) ← propose-edge(G, Δ)
            if edit-edge commits and Apply(G, Δ) passes invariants then
                G ← Apply(G, Δ);  append Δ to H {reversible by recency}
            end if
        end if
    end while
end for
return G, H
```

Cold-start configuration.

- Embedder: `text-embedding-3-large`
- LLM classifier: `gpt-5-nano` for pair classification
- Candidate pair threshold: adaptive (mean cosine similarity $+1\sigma$, clipped to $[0.35, 0.75]$)
- Maximum candidates per node: 12, with a minimum floor of 3

B Failure Analysis

B.1 ALFWorld Failure Modes

Among the 11 failures in the 30-task pilot evaluation, we identify three categories:

Tight loops (8/11). The agent enters a cycle of repeated actions that produce "Nothing happens" observations. The most common pattern is attempting to interact with an object that is not in the current room, without recognizing the need to navigate elsewhere. The graph provides the correct procedural skills, but the agent fails at the grounding level (mapping skill instructions to valid environment actions).

Meta-mode confusion (1/11). A single failure (task index 17) exhibited a pathological pattern where the model's internal reasoning tokens leaked into the action output, causing the agent to hallucinate observations and believe the task was complete. This is a model-specific artifact (MiniMax `</think>` tag leakage) rather than a skill-selection failure.

Step budget exhaustion (2/11). The agent explores correctly but exhausts the 30-step budget before completing all required subtasks. These tasks involve longer action sequences (e.g., cleaning an object requires finding it, picking it up, navigating to a sink, using the sink, then navigating to the target location).

C Edge Type Operational Definitions

We provide expanded operational definitions for each edge type, including the counterfactual tests that would ideally validate them:

`depends_on`(A, B)**.** Removing skill B from the agent's available set drops the success rate of A by a measurable margin. This captures prerequisite relationships: B provides setup, context, or intermediate results that A requires. Transitivity holds: if A depends on B and B depends on C, then A implicitly depends on C.

`composes_with`(A, B)**.** The joint success rate of tasks where both A and B are selected exceeds $\max(\text{success}(A), \text{success}(B))$ by a measurable margin. This captures synergistic combinations where skills complement each other. Symmetry holds. Transitivity does not, because composability is pairwise rather than chainable.

`similar_to`(A, B)**.** Skills A and B achieve comparable success rates on the same task distribution and can substitute for each other. Co-selecting both wastes context but does not cause harm. This is the weakest relationship type: it signals redundancy rather than interaction.

`conflicts_with`(A, B)**.** Co-selecting skills A and B causes a measurable drop in success rate compared to selecting either alone. This captures genuine interference: resource contention, contradictory instructions, state contamination, or protocol conflicts. Symmetry holds. Transitivity does not.

`specializes`(D, A)**.** Skill D achieves higher success than A on a specific subdomain of A's applicability, while inheriting A's core functionality. When D is available and the task falls in its subdomain, D should be preferred over A. Transitivity holds within specialization hierarchies.

D Reproducibility Details

Search budgets. All `search` calls use $K = 5$ vector matches and BFS neighbor depth $D = 2$. The agent may issue any number of `search` calls per episode and chooses query strings on its own schedule.

Models and embeddings. Agent runs use MiniMax-M2.7 (via the official MiniMax Anthropic-compatible endpoint) and `gpt-5.2-codex` (via the OpenAI Codex CLI with `reasoning_effort=high`). Cold-start pair classification uses `gpt-5-nano`. Query embedding at runtime uses `text-embedding-3-large`, matching each GoS workspace's embedding configuration.

Step and attempt budgets. ALFWorld episodes are capped at 30 steps with $n_{\text{attempts}} = 2$ following Liu et al. (2026) §4.1. SkillsBench uses $n_{\text{attempts}} = 1$ for the MiniMax cross-scale retrieval replay and $n_{\text{attempts}} = 2$ for main task-reward results; the `gpt-5.2-codex` retrieval row replays the codex agent's queries under the same $n_{\text{attempts}} = 2$ budget.

Online edit logging. Each accepted `edit-edge` writes an entry with `origin` (cold-start vs. online), `reason` (natural-language justification), and `task_id` (the episode that produced the supporting evidence) into the graph's edge metadata.

Cold-start cost. At scale 1000 the cold-start construction takes approximately 200 `gpt-5-nano` calls in batched bucket mode (one call per anchor skill, classifying all candidate pairs in that bucket), plus one embedding call per skill for each of the two views (e_{self} and e_{needs}).

E Worked Example Subgraph

Edges are unweighted (present or absent); the LLM decides whether to add, remove, or retype an edge based on execution evidence, not statistical accumulation. Table 3 lists the runtime role of each type; the main-text Figure 2 shows a worked subgraph exercising all five.

Table 3: Edge type taxonomy; runtime roles.

Type	Runtime role
`depends_on`	Selecting A pulls in B as prerequisite
`composes_with`	Surfaces composition opportunities
`similar_to`	De-duplication: select one, not both
`conflicts_with`	Prune: selecting A excludes B
`specializes`	Prefer specialized D over general A

F Search Interface Field Semantics

Matches. Cosine similarity is computed between the query embedding and each node's pre-computed embedding; the top-K are returned, ranked by similarity.

Neighbors. From the match set, BFS proceeds along walkable types (`specializes`, `composes_with`, `depends_on`, `similar_to`) up to depth D, traversed bidirectionally so the agent sees both ends of a relationship (for `depends_on`, both the prerequisite and any skill requiring the match as setup); the typed label travels with the edge. Each neighbor records its shortest distance from a match, predecessor, and edge type. `conflicts_with` is not traversed. Skills already matched are excluded.

Conflicts. All `conflicts_with` edges incident to any match within one hop are returned separately. They have opposite polarity from neighbors: skills that should *not* be co-selected. No transitive expansion (conflicts-of-conflicts are undefined).

Properties. (i) The agent issues queries on its own schedule and rephrases as understanding evolves, rather than receiving one episode-start bundle. (ii) The three-field decomposition lets the agent drop a whole field under tight budget rather than truncating an opaque concatenation. (iii) Edges carry an explicit type and reason, so the agent's trace documents selection/exclusion. After `search`, the agent issues `show` for chosen candidates; the decision is unconstrained (all matches, only typed neighbors, or skip the graph).

G Online-Edit Protocol Detail

The protocol separates inspection from mutation. `propose-edge` performs a dry-run returning what the change would do and surfacing all existing edges and recent history entries for the same skill pair, so the agent sees what prior episodes established before proceeding. `edit-edge` commits, appending a history entry with task ID, natural-language reason, and timestamp. The per-edit `reason` and `task_id` log preserve auditability without gating the edit, placing graph-quality responsibility on the same agent that consumes it. The three commit-time invariants (acyclicity of the `depends_on`/`specializes` backbone; non-contradiction, meaning no `conflicts_with` edge on a pair that already carries a positive surfacing edge such as `depends_on`, `specializes`, `composes_with`, or `similar_to`; and rollback by recency or task ID via the append-only log) are enforced on both `propose-edge` and `edit-edge`.

H Cold-Start Pipeline Detail

The first view $e_{\text{self}}(v)$ embeds v's identifier, description, and a body preview. The second view $e_{\text{needs}}(v)$ prompts an LLM to imagine two or three concrete tasks invoking v in a longer chain and summarize the shared prerequisites in one sentence, which is embedded; skills with no clear shared prerequisite emit `self-contained` and are excluded from the second view. Candidate pairs come from two rankings: $\cos(e_{\text{self}}(i), e_{\text{self}}(j))$ over all pairs, and $\cos(e_{\text{needs}}(i), e_{\text{self}}(j))$ for each i with non-empty needs. A pair is kept when its cosine exceeds an adaptive threshold $\max(0.35, \min(0.75, \mu + \sigma))$ (μ, σ the mean/SD of that ranking's similarities). The two-view design exists because e_{self} alone misses cross-functional pairs: a cooling skill and a pickup skill are distant under e_{self}, but the cooling skill's needs text ("object in hand") bridges to the pickup skill's self-description. The candidate union is bucketed by anchor (lower-index skill) and sent to an LLM in one call per bucket assigning `similar_to`, `specializes`, `depends_on`, `composes_with`, or `none`; `none` pairs are discarded and the rest become cold-start edges carrying the LLM's reason.