SKILLWORLD: WORLD-INDEXED SKILL GRAPHS WITH GOVERNED SELF-EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Composing skills from growing libraries for task execution requires reasoning about dependencies, alternatives, and conflicts among them. The information needed to infer these relations is distributed across skill descriptions, and their applicability may vary with the task and execution context. This calls for a reusable skill graph that supports context-dependent interpretation while remaining consistent as the underlying skill knowledge evolves. We present SKILLWORLD, which derives skill relations from typed skill descriptions and a shared ontology to provide a relational knowledge layer over large skill libraries. At runtime, it evaluates relation applicability in the current task and execution context. As the skill library grows, SKILLWORLD evolves the shared ontology to incorporate new skill knowledge and re-derives relations to maintain consistency with the updated model. On SkillsBench, SKILLWORLD with evolved static graphs outperforms graph-based and retrieval baselines, achieving gains of 11.1–15.3 percentage points in task success rate over the strongest baseline for each evaluated model. On ALFWorld, the full system produces 43.5–72.4% fewer invalid environment actions than BM25 skill retrieval, while achieving 85.1–85.8% task success within 15 environment actions across two models. We also deploy and evaluate SKILLWORLD in a large company's internal services. From the deployment traces, we select 36 tasks that are difficult to complete without skills. SKILLWORLD achieves a mean success rate of 72.2% over three runs.

1 INTRODUCTION

Skill libraries give language-model agents reusable procedures for carrying out tasks, from executable routines to workflows recovered from prior experience (Wang et al., 2023; Yuan et al., 2024; Wang et al., 2025). Using several skills together requires deciding how they fit: which skill supplies an input or establishes a condition for another, which skills offer alternatives, and which combinations interfere. These decisions depend on the environment in which the skills are used. For example, a request to run a project's tests can retrieve environment preparation as a potential supporter. Missing dependencies make that support useful now; a ready environment removes the need for preparation while leaving the structural relation intact. The query and skill library remain the same; the current state determines how the candidate should be used. A reusable skill library therefore needs an account of both how its procedures relate and when those relations matter during execution.

Retrieval and graph-based methods address different parts of this problem. Tool retrieval identifies relevant APIs (Qin et al., 2024), while tool and skill graphs make relational structure available for navigation and selection (Liu et al., 2024; 2026a). Recent systems also maintain execution-backed graph edits or repair composed skill graphs (Bai et al., 2026; Xia et al., 2026); feedback-driven agents refine their decisions across trials (Shinn et al., 2023) and hone reusable skills through practice (Zheng et al., 2025). These developments bring retrieval, composition, and adaptation together. For a graph shared across a library, their combination raises a further requirement: the reasons for storing a relation must remain connected to the conditions under which it is used. Whether a dependency observed in one execution is useful in another depends on the state and resource bindings encountered there. Likewise, correcting an assertion about a skill can affect several relations that rely on it. Maintaining such a graph requires treating its edges as consequences of an explicit model that can be re-evaluated and revised.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We study this problem through a world-indexed skill graph: a reusable model of potential relations together with an operational view conditioned on the current state, resource bindings, and goal. Our focus is the relational model shared by a skill library over repeated executions. The key idea is to derive relations from a common description of skill behavior, specialize them where they are used, and revise that description when evidence supports a correction. We call a graph coherent when its edges follow common rules over one pinned model, current when its operational relations are re-evaluated against the execution context, and corrigible when evidence can revise the assertions supporting those relations. These properties connect construction, use, and maintenance of the same graph.

Three difficulties arise in making this connection. First, skill artifacts describe capabilities, inputs, conditions, and effects using different vocabularies and interface conventions. Deriving relations across them requires a shared semantic representation, with enough provenance to locate the assumptions behind an edge. Second, execution provides partial observations. Unobserved conditions and unresolved resource bindings require an explicit unknown status. The operational view must preserve this uncertainty while using the available information. Third, extracted skill assertions can indicate gaps or conflicts in the shared vocabulary, and compilation can expose inconsistent structure. These findings must be tied to specific model assertions and checked before they support an edit. The revised model must then satisfy the shared constraints before its derived relations become available.

We present SKILLWORLD, which connects these three stages through an explicit, recompilable model. Each skill artifact is interpreted into a reusable typed manifest. Fixed Datalog rules executed by Soufflé (Jordan et al., 2016) combine the manifests with a shared ontology to derive five relation families, retaining the facts and witnesses behind each edge and reusing unchanged manifests across updates. At runtime, a deterministic, LLM-free projection evaluates the potential graph against the current context and returns confirmed, unknown, and rejected relations with typed reasons. Candidate discovery traverses the potential relations from either endpoint and packages skill descriptions, relation evidence, and state annotations as structured context for language-model selection. In the testing example, environment preparation remains a candidate alongside the requested testing skill; its current status distinguishes immediate support from an inactive preparation step. Between releases, evidence extracted from skill artifacts and manifests, together with compilation diagnostics, guides revision of model assertions. Exact proposals form a candidate menu for LLM review. After selection, a deterministic source-consistency check verifies source support and detects conflicting claims for supported vocabulary edits before write authorization. Authorized proposals stage the corresponding ontology entries. Selective re-extraction refreshes affected manifests; recompilation and validation precede atomic publication of a successor release. Projection interprets the active release in the current execution context, while evolution revises the shared model across releases. Both preserve a traceable connection between a relation and its supporting assumptions.

Our contributions address three requirements for a reusable skill graph:

- **Coherent: shared semantics for skill relations.** We provide a unified basis for interpreting and revising the graph by formulating relations as derivable claims over reusable typed skill descriptions and a shared ontology, with common inference rules and explicit support for each edge (Section 3.1.2).
- **Current: world-indexed relation applicability.** We distinguish a relation's potential existence from its applicability in a particular execution context. A three-valued semantics evaluates relations against the current state, resource bindings, and goal while preserving uncertainty under partial observations (Section 3.1.3).
- **Corrigible: evidence-guided revision of shared knowledge.** We make the assertions supporting skill relations revisable through source-verified edits. Re-deriving their consequences propagates corrections across the graph while preserving evidence traceability and model well-formedness (Section 3.2).

2 RELATED WORK

Typed operators and compiled composition. Classical planning used action operators (Fikes & Nilsson, 1971), and temporal abstraction introduced reusable options (Sutton et al., 1999). Graphplan built layered planning graphs (Blum & Furst, 1997), while learned symbolic models used skill

initiation and effect sets for planning (Konidaris et al., 2018). OWL-S and type-directed synthesis composed declared service or API interfaces (Martin et al., 2004; Sirin et al., 2004; Feng et al., 2017). Deep Skill Graphs learned graphs for exploration and planning (Bagaria et al., 2021); SkillNet organized skills through an ontology and relation graph (Liang et al., 2026). SKILLWORLD compiles a provenance-carrying relation model per library release.

Learned skill and tool graphs. ToolNet updated tool-transition weights from execution trajectories (Liu et al., 2024); ControlLLM searched a pre-built dependency graph (Liu et al., 2023). Graph-of-Skills built an offline retrieval graph, and SkillTrace traversed query, similarity, and dependency levels (Liu et al., 2026a; Yao et al., 2026). SE-GoS evolved retrieval-graph topology, edge weights, and retrieval-facing descriptions from execution traces while preserving the GoS retrieval procedure and skill content (Fu et al., 2026). SkillDAG added execution-backed edits to a persistent graph; GraSP repaired a task-local DAG; and SkillComposer selected and ordered skills for a task (Bai et al., 2026; Xia et al., 2026; Zhao et al., 2026). GTool conditioned tool planning on request-tool graphs (Chen et al., 2025). Other methods paired mined transition priors with learned ordering (Liu & Li, 2026), jointly evolved skill graphs and policies (Li et al., 2026b), or composed executable skill networks (Shi et al., 2026). SKILLWORLD derives relations from a shared semantic model, evaluates their applicability in the current execution context, and revises their supporting assertions using artifact evidence and compilation diagnostics.

Context-indexed relational views. Description logics distinguish schemas from instances (Baader et al., 2007), and contextual repositories organize knowledge by context (Serafini & Homola, 2012). Partial evaluation specializes programs using known inputs (Leuschel & Bruynooghe, 2002); incremental view maintenance updates derived relations (Gupta et al., 1993), and provenance records derivation dependencies (Green et al., 2007). Environmental grounding also informs language-guided action selection and policy generation in SayCan and Code as Policies (Ahn et al., 2022; Liang et al., 2023). Agent systems conditioned hierarchies, tool frontiers, support graphs, or state–action graphs (Hao et al., 2026; Babu & Iyer, 2026; Zhai et al., 2026; Tang et al., 2025); SKILLWORLD evaluates the library's candidate relations in a typed read-only step view that guides skill retrieval.

Ontology maintenance and agent evolution. Ontology evolution studied knowledge revision (Noy & Klein, 2004; Flouris et al., 2008); database repair formalized minimal repairs and consistent query answers (Arenas et al., 1999). Agents refined behavior through verbal feedback and self-discovered skills (Shinn et al., 2023; Zheng et al., 2025). Recent systems used skill contracts (Shen et al., 2026; Song et al., 2026), applied checked edits and rollback (Bai et al., 2026; Zhang et al., 2026b), and gated updates through evaluation or verification (Shawn, 2026; Banerjee et al., 2026; Huang & Huang, 2025; Zhang, 2026). A lifecycle survey organized these mechanisms across evolving skill libraries (Li, 2026). Trace-conditioned and co-evolving systems coupled revisions to verification or calibrated evidence (Liu et al., 2026b; Zhang et al., 2026a; Tang et al., 2026). SKILLWORLD uses typed artifact evidence and compilation diagnostics to revise the shared manifest and ontology model, then recompiles affected relations and publishes a checked successor release.

3 METHODOLOGY

SKILLWORLD combines world-indexed skill graphs with governed self-evolution. Figure 1 shows how shared semantics derive relations, current state guides skill selection and use, and artifact evidence with compilation diagnostics supports revision of the model.

3.1 WORLD-INDEXED SKILL GRAPHS

SKILLWORLD separates structural relatedness, current applicability, and candidate selection. Compilation derives a shared potential graph from skill interfaces. Projection annotates each relation in the current world. Retrieval packages candidates, their structural relations, and these annotations as structured context for language-model selection.

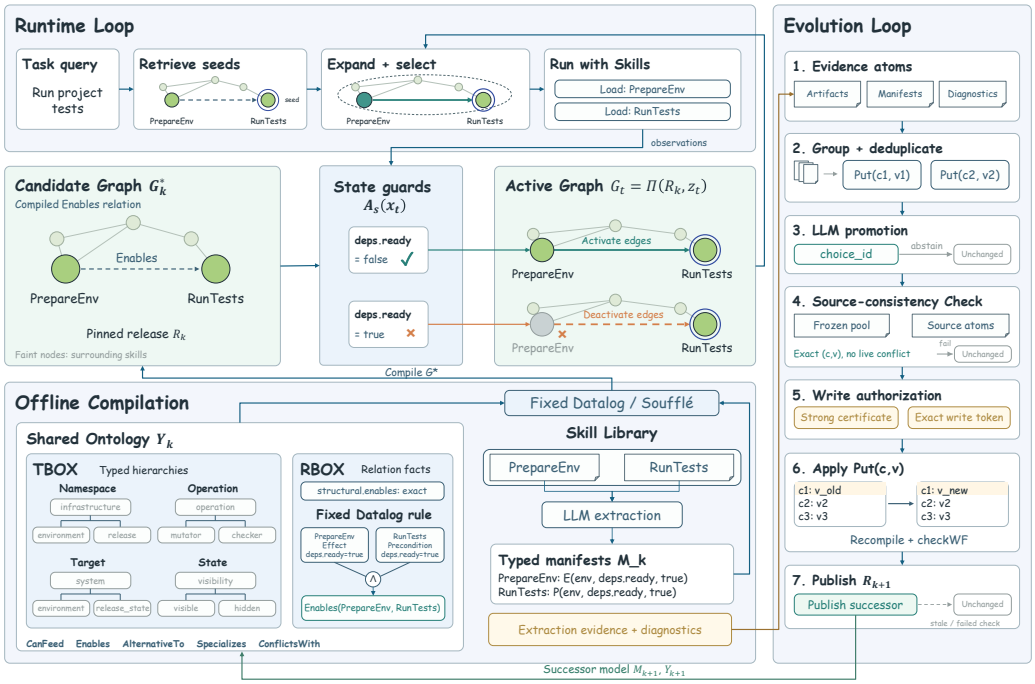

Figure 1: Offline, typed manifests M_k and ontology Y_k feed fixed Datalog rules to derive G_k^*. At runtime, retrieval expands `RunTests` with `PrepareEnv`, and an LLM selects skills using projected relation context. In the example, the `Enables` edge is active for `deps.ready=false` and inactive for `deps.ready=true`. Evolution groups artifact and manifest evidence with diagnostics into $\mathrm{Put}(c, v)$ proposals. LLM selection and native Lean source-consistency checks precede authorization, recompilation, well-formedness checking, and publication; abstention or failed checks retain the current release.

3.1.1 Skills, Manifests, and the Ontology

A skill contains instructions, scripts, and tool bindings. SkillWorld normalizes its declared interface into

$$m_s = (c_s, I_s, O_s, P_s, E_s, K_s),$$

where c_s is a typed capability, I_s and O_s are input and output ports, P_s and E_s are preconditions and effects, and K_s records constraints such as destructive behavior. All semantics operate on m_s; "applicable" means that its declared hard preconditions hold.

Manifests are interpreted against an ontology $Y = (N, C, T, D, \mathcal{A})$, where N, C, T are namespaces, operations, and resource targets; D defines state dimensions and values; and $\mathcal{A}$ contains relation axioms.

The terminological box (TBox) defines the vocabularies and hierarchies of N, C, T, D, together with resource compatibility and valid state values. The role box (RBox) represents $\mathcal{A}$ as relation declarations, matching parameters, evidence requirements, and contradiction facts. These ontology entries are supplied as facts to the fixed Datalog rules, which combine them with skill-specific manifest assertions to derive relations.

Hard conditions ground into atoms $(\text{subject}, d, v)$, where $d \in D$, the subject names a resource or port role, and v is a valid value for d. SkillWorld retains non-normalizable claims as typed unknowns and excludes them as grounds for derivation or action. Source spans make each interpretation traceable; repair writes only to a recompiled successor release.

3.1.2 COHERENCE BY COMPILATION: THE POTENTIAL GRAPH

Given release-pinned manifests $M_k = \{m_s\}_{s \in S}$ and ontology Y_k, a fixed Datalog program executed by Soufflé (Jordan et al., 2016) combines canonical manifest, TBox, and RBox facts to produce an immutable graph of potential relations,

$$G_k^\star = \mathcal{C}(M_k, Y_k).$$

The published object is the immutable release

$$\mathcal{R}_k = (M_k, Y_k, G_k^\star, q_k),$$

where q_k is the administrative edge-quarantine map. A potential edge records a structural witness shared across runtime contexts, independently of current endpoint applicability. Its witness can cover a subset of the consumer's resource or precondition requirements.

The graph contains five relation types: `CanFeed`, `Enables`, `AlternativeTo`, `Specializes`, and `ConflictsWith`. The first four support candidate discovery from either endpoint; conflicts provide constraint context. For skills with valid primary capabilities, the configured exact-match rule derives `Enables` from a shared normalized effect and precondition:

$$\frac{a \neq b \qquad (\tau, d, v) \in E_a \qquad (\tau, d, v) \in P_b}{(a, \mathsf{Enables}, b) \in G_k^\star}. \tag{1}$$

Here τ is a subject target, and v is a valid value of state dimension d under Y_k. In Figure 1, the effect `deps.ready=true` of `PrepareEnv` matches a precondition of `RunTests`. Each derived edge retains its rule, source facts, and matched witness. Appendix B gives the other relation schemas.

The manifest is the unit of semantic interpretation; an edge is a derived artifact. With the ontology unchanged, adding a skill reuses existing manifests and requires extraction only for the added artifact. A revised ontology assertion can change several relations at once because the same rules interpret the shared model across the library. Provenance identifies which manifest facts and ontology axioms produced each affected edge, providing the coordinates that later inspection and repair can revisit.

3.1.3 CURRENCY BY PROJECTION: THE ANNOTATED VIEW

Context and view. At episode step t, the runtime context is $z_t = (x_t, b_t, g_t)$, where x_t is finite grounded state, b_t partially binds ports and condition roles to resources, and g_t identifies the goal and candidate skills. These step-scoped facts partially evaluate the published graph:

$$G_t = \Pi(\mathcal{R}_k, z_t) = \{(e, \mathsf{projectPart}_k(e, z_t)) : e \in G_k^\star\},$$

where

$$\mathsf{projectPart}_k(e, z) = \begin{cases} \mathsf{rejected}, & q_k(e), \\ \mathsf{part}(\phi_{r(e)}(e, z)), & \text{otherwise.} \end{cases}$$

Here ϕ_r is the activation predicate of relation type r evaluated in Strong–Kleene logic (Kleene, 1952) $\mathbb{K}_3 = \{\mathsf{F}, \mathsf{U}, \mathsf{T}\}$; part maps these values to rejected, unknown, and confirmed; and $q_k(e)$ is the release's administrative quarantine bit. Quarantine assigns an administrative rejection independently of the truth of ϕ_r. Thus G_t annotates every candidate, with $G_t^+ = \{e \in G_k^\star : \mathsf{projectPart}_k(e, z_t) = \mathsf{confirmed}\}$. Projection updates this view while preserving $\mathcal{R}_k$.

For an `Enables` edge $a \to b$, the activation predicate is

$$\phi_{\mathsf{Enables}}(a, b, z_t) = A_a(z_t) \wedge \neg A_b(z_t) \wedge A_b(T_a(z_t)). \tag{2}$$

A_s evaluates the grounded hard preconditions of skill s in $\mathbb{K}_3$, and T_s applies its declared hard effects; unresolved post-state evaluations yield U. Activation therefore requires an applicable source, a blocked target, and a source transition that satisfies all of the target's hard preconditions. In the figure's single-condition example, an applicable `PrepareEnv` enables `RunTests` while `deps.ready=false`; the relation becomes inactive once `deps.ready=true`.

A persistent Soufflé service evaluates the finite, non-recursive activation predicates over $G_k^\star$, returning the complete three-way partition and typed reasons.

3.1.4 STRUCTURED CONTEXT FOR SKILL SELECTION

A retriever supplies query-relevant seed skills. SKILLWORLD expands them along the four candidate-expanding relations from either endpoint under shared candidate and hop budgets. Confirmed, unknown, and semantically rejected edges remain available for discovery; quarantined edges and endpoints are excluded. `ConflictsWith` contributes constraint context without expanding the pool.

The resulting context contains skill identifiers, summaries and declared interfaces, typed relations with their directions and supporting evidence, and, when available, current state annotations from projection. The language model uses this context to select skills by task relevance and their joint usefulness. Blocked skills remain available for planning; execution requires their hard preconditions to hold. The selected documents are then loaded for the agent. Subsequent calls refresh the context with updated observations and bindings. Appendix B specifies the context records and selection interface.

Each call returns its release identifier; Appendix C specifies snapshot isolation and episode-level pinning.

3.2 GOVERNED SELF-EVOLUTION

SKILLWORLD stages source-verified ontology edits, selectively refreshes manifests, and validates their compiled relations before publication. Algorithm 1 gives the single-edit version of this offline loop. The finite menu $\mathcal{Q}$ tracks unprocessed edits; frozen evidence $\mathcal{E}$ and source manifests $\overline{M}$ supply the source-consistency check.

Algorithm 1 Evidence-guided evolution with re-extraction (single-edit case)

Input: Active release $\mathcal{R}_0$, skill artifacts $\mathcal{S}$, budgets $\mathcal{B}$
Output: Active release $\mathcal{R}$ and termination status

$\mathcal{R} \leftarrow \mathcal{R}_0$; $M \leftarrow \textsc{Manifests}(\mathcal{R}_0)$
$(\mathcal{E}, \overline{M}) \leftarrow \textsc{FreezeEvidence}(\mathcal{S}, \mathcal{R})$
$\mathcal{Q} \leftarrow \textsc{GroupAndDeduplicate}(\mathcal{E})$
while $\mathcal{Q} \neq \varnothing$ **do**
  $u \leftarrow \textsc{LLMPromote}(\mathcal{Q}, \mathcal{R})$
  if $u = \bot$ **then return** $(\mathcal{R}, \mathsf{abstained})$
  $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{u\}$
  $\mathcal{E}_{\mathrm{src}} \leftarrow \textsc{ReparseSources}(\overline{M}, u)$
  if $\textsc{SourceConsistent}(u, \mathcal{E}, \mathcal{E}_{\mathrm{src}}) \wedge \textsc{PlanValid}(u, \mathcal{R})$ **then**
    $Y' \leftarrow \textsc{StageAuthorizedPut}(\mathcal{R}, u)$
    $(A, d) \leftarrow \textsc{AffectedSkills}(\mathcal{S}, M, u, \mathcal{B})$
    if $d \neq \mathsf{accepted}$ **then return** $(\mathcal{R}, d)$
    $(M', d) \leftarrow \textsc{ReextractAndReview}(\mathcal{S}_A, Y', M, \mathcal{B})$
    if $d \neq \mathsf{accepted}$ **then return** $(\mathcal{R}, d)$
    $\mathcal{R}' \leftarrow \textsc{BuildSuccessor}(\mathcal{R}, M', Y', \mathcal{C}(M', Y'))$
    if $\textsc{CheckedInput}(\mathcal{R}') \wedge \mathsf{checkWF}(\mathcal{R}') \wedge \textsc{Publish}(\mathcal{R}, \mathcal{R}')$ **then**
      $\mathcal{R} \leftarrow \mathcal{R}'$; $M \leftarrow M'$
      $(\mathcal{E}, \overline{M}) \leftarrow \textsc{FreezeEvidence}(\mathcal{S}, \mathcal{R})$
      $\mathcal{Q} \leftarrow \textsc{GroupAndDeduplicate}(\mathcal{E})$
    else return $(\mathcal{R}, \mathsf{blocked})$
return $(\mathcal{R}, \mathsf{menu_empty})$

3.2.1 EVIDENCE-GROUNDED CANDIDATE GENERATION

Evidence $\mathcal{E}$ is drawn from skill artifacts, structured manifests, and compilation diagnostics that identify gaps or discrepancies in the model. Diagnostics identify assertions to inspect; an evidence atom specifying a concrete coordinate c and proposed value v introduces a candidate edit $\mathsf{Put}(c, v)$. Each candidate retains the current value, supporting source text, risk labels, and support counts. Equivalent proposals are grouped by semantic content, with counts deduplicated across source roots and declared provenance groups. Conflicting values remain separate candidates with explicit references to their supporting evidence, forming a finite menu for review.

3.2.2 LLM SELECTION AND SOURCE-CONSISTENCY CHECK

Given this menu, the LLM promotion operator reviews the proposed values against current assertions and supporting sources, considering whether the evidence supports each assignment and its scope. Each invocation selects existing candidate identifiers or abstains. Batched selection rejects overlapping coordinate writes and combines the selected edits into one plan.

The verifier performs a deterministic *source-consistency check* for structured manifest proposals of namespace, operation-class, target, and state edits. For each selected edit in this scope, it reconstructs the candidate from the frozen evidence pool. It uses skill identifiers in the supporting evidence to locate the frozen source manifests. Their structured proposals are reparsed into source evidence atoms. Acceptance requires an active supporting atom that appears identically in both collections and exactly supports the selected coordinate and complete value. All active claims for that coordinate in either collection must agree semantically; verifier-accepted retractions determine which claims remain active. A native Lean checker enforces these conditions. The system records the source evidence for each accepted edit.

3.2.3 SELECTIVE RE-EXTRACTION AND CHECKED PUBLICATION

Validated edits enter a staged ontology Y'. AFFECTEDSKILLS supplements direct dependencies with source-reviewed retrieval, including skills without exact proposals. REEXTRACTANDREVIEW refreshes selected manifests and reviews semantic changes within token and extraction budgets, reusing other manifests. CHECKEDINPUT verifies provenance, membership, and the rebuilt graph. Publication requires these checks and checkWF on the successor $(M', Y', \mathcal{C}(M', Y'))$. Refreshed evidence supplies the next menu; an empty menu or explicit abstention ends evolution. Unresolved reviews or exhausted budgets block the round. Appendix D details selection, review, and the Lean 4 source, refresh, and publication contracts.

4 EVALUATION

We evaluate SKILLWORLD through task success and invalid environment actions. SkillsBench assesses how a coherent semantic model (Section 3.1.2) and governed self-evolution (Section 3.2) improve skill-graph quality and model task execution. We compare base and evolved graphs with retrieval and graph-based baselines, holding each graph fixed during task execution. ALFWorld tests whether observation-conditioned retrieval improves task completion and reduces invalid actions while keeping the ontology, manifests, and potential graph frozen (Section 3.1.3). We additionally evaluate difficult tasks curated from internal-service deployment trajectories, using the deployed ontology release with all SKILLWORLD components enabled (Section 4.3).

4.1 TASK EXECUTION ON SKILLSBENCH

Setup. SkillsBench (Li et al., 2026a) supplies document-scale skills and task verifiers. We evaluate Qwen3.8-Max and Claude Opus 4.8 on 72 of 87 tasks (15 excluded after three infrastructure-error attempts each; Appendix E). From the 500-skill library, BM25 and BM25 with LLM reranking consider the top 20 lexical matches. SKILLWORLD expands eight BM25 seeds along four discovery relations to at most 20 candidates, then ranks with five relation groups. Each method gives the agent its top five skills. Baselines include BM25, BM25 with LLM reranking, vector retrieval using `text-embedding-3-large`, a co-use graph excluding the current task's contribution, GoS (Liu et al., 2026a), SkillDAG (Bai et al., 2026), and no skill. Each model–method configuration runs each task three times using `claude-agent-acp`. We report mean passed counts and success rates (SR) across the three runs, with SR computed over all 72 tasks in each run. Base and evolved graphs remain fixed during task execution, without runtime state projection.

Evolution. Evolution follows Algorithm 1, with at most one additional review candidate per vocabulary coordinate and explicit refresh-token and extraction limits (Appendix D). Each action counts one ontology edit incorporated into a published release. Qwen completed 51 rounds (379 actions, 412 manifest re-extractions), stopping with an empty executable menu at the round-52 check. Opus completed 41 rounds (168 actions, 194 manifest re-extractions) and stopped after explicitly

abstaining across all candidate batches. Re-extraction and recompilation precede task execution. Appendix G traces one cycle through re-extraction and graph reuse.

Table 1: SkillsBench task execution on 72 tasks. Passed counts and success rates (SR) are averaged over three runs; higher is better. Bold marks the best value per column.

	Qwen3.8-Max		Claude Opus 4.8	
Method	Mean passed	Mean SR (%)	Mean passed	Mean SR (%)
No skill	14/72	19.4	22/72	30.6
BM25	25/72	34.7	37/72	51.4
BM25 + LLM reranking	25/72	34.7	37/72	51.4
Vector	27/72	37.5	38/72	52.8
BM25 + Co-use Graph	24/72	33.3	40/72	55.6
Graph of Skills (GoS)	27/72	37.5	36/72	50.0
SkillDAG	28/72	38.9	37/72	51.4
SKILLWORLD (base)	26/72	36.1	39/72	54.2
SKILLWORLD (evolved)	**39/72**	**54.2**	**48/72**	**66.7**

Results. Relative to the base graph, success rate with evolved SKILLWORLD increases from 36.1% to 54.2% for Qwen and from 54.2% to 66.7% for Opus, gains of 18.1 and 12.5 percentage points, respectively. Mean passed counts are 39/72 for Qwen and 48/72 for Opus (Table 1). Its success rate is higher than the strongest execution baseline by 15.3 points for Qwen (SkillDAG) and 11.1 points for Opus (BM25 + Co-use Graph).

4.2 STATE-CONDITIONED RETRIEVAL ON ALFWORLD

We use ALFWorld (Shridhar et al., 2021) to test state-conditioned retrieval and relation-level projection under a frozen skill graph.

Setup and metrics. The 500-skill ALFWorld library replaces unrelated SkillsBench skills with an equal number of ALFWorld-specific skills. Qwen3.8-27B and DeepSeek V4.1 Flash use fixed model parameters. Within each model, all ablations retain the same skills, ontology, manifests, and potential graph throughout evaluation. We use 134 public unseen games with three repeats per game (seeds 42, 43, and 44), giving 402 episodes per condition (repeat-level results in Appendix F). Each episode permits at most 15 environment actions. We report task completion within 15 environment actions (SR@15) and invalid environment actions, whose observations contain `nothing happens` (case-insensitive). Invalid-action counts are summed over successful and failed episodes; model-response format errors, skill-document requests, and network failures are excluded.

State-context comparison and baselines. LLMs extract symbolic state from public observations and bind observed resources to skill roles. Full uses the current symbolic state and resource bindings for deterministic projection. Static-default uses an LLM to infer state from the initial public observation and keeps that projection context fixed throughout the episode. Static-random independently samples each state attribute from its legal values and keeps the resulting projection context fixed throughout the episode. No relation labels retains Full's dynamic state, skill applicability, candidate-pool construction, models, and budgets. It ablates relation annotations together with their use in candidate ordering and context selection. In the graph-based ALFWorld conditions, the actor may request documents from the full candidate catalog without a separate LLM reranking call. Actors receive subsequent public observations in all conditions. System baselines use SkillDAG (Bai et al., 2026), GoS (Liu et al., 2026a), BM25 retrieval over skill documents (No graph), or no skill library (No skills). We use the upstream SkillDAG implementation (commit `2bd7baff`) with its own prompts, raw public observations, vector retrieval with typed-graph expansion, and online graph updates under the same task and resource budgets. Its graph is shared across the 134 games within each model/seed queue and reset between seeds.

Effect of current state and relation-level context. Full achieves 85.07% SR@15 with Qwen and 85.82% with DeepSeek (Table 2). Its SR@15 is higher than Static-default by 2.99 and 3.48

Table 2: ALFWorld results over 402 episodes per condition (134 unseen games, three repeats). SR@15 is completion within 15 environment actions (↑); Invalid counts `Nothing happens` across successful and failed episodes (↓). Bold marks column bests. SkillWorld freezes its ontology and graph; SkillDAG updates online.

	Qwen3.8-27B		DeepSeek V4.1 Flash	
Condition	SR@15 (%)	Invalid ↓	SR@15 (%)	Invalid ↓
Full	**85.07**	**74**	**85.82**	**51**
No relation labels	75.12	115	79.85	162
Static-default	82.09	94	82.34	80
Static-random	80.85	104	79.10	85
SkillDAG	81.09	147	84.33	170
Graph of Skills (GoS)	69.90	316	74.88	113
No graph (BM25)	75.37	131	80.60	185
No skills	69.90	412	64.43	468

percentage points, and higher than Static-random by 4.23 and 6.72 points, respectively; differences are computed before rounding. No relation labels achieves 75.12% SR@15 for Qwen and 79.85% for DeepSeek, respectively 9.95 and 5.97 percentage points below Full. Invalid-action counts increase from 74 to 115 for Qwen and from 51 to 162 for DeepSeek. These results support the combined contribution of relation annotations and candidate organization beyond individual skill applicability.

Comparison with system baselines. Full improves task success over SkillDAG by 3.98 and 1.49 percentage points and reduces invalid-action counts by 49.7% and 70.0%. It also achieves higher task success than GoS, BM25, and no skills with both models; relative to BM25, invalid actions decrease by 43.5% and 72.4%.

4.3 Task Execution on Internal Services

Setup. We deployed SkillWorld as a service on an internal platform for complex software development, deployment, and documentation, and collected execution trajectories. From these, we curated and annotated 36 tasks that were difficult to complete without skills, then re-executed them with Qwen3.8-27B. We compared no skill, BM25, GoS, SkillDAG, and SkillWorld; skill-based conditions used the same 903-skill internal library. A human-annotated oracle provided manually selected skills as a reference. Each condition ran the same 36 tasks three times. We report mean completed counts and success rates (SR), with SR computed over all 36 tasks in each run.

Table 3: Mean Qwen3.8-27B task success over three runs of the same 36 internal-service tasks with a 903-skill library. Bold marks the best non-oracle result; the oracle uses human-selected skills.

Method	No skill	BM25	GoS	SkillDAG	SkillWorld	Oracle
Mean completed	6/36	13/36	11/36	24/36	**26/36**	30/36
Mean SR (%) ↑	16.7	36.1	30.6	66.7	**72.2**	83.3

Results. SkillWorld completed 26 of 36 tasks on average (72.2% mean SR), the highest non-oracle result (Table 3). It averaged 20 more completed tasks than no skill (55.6 points) and two more than SkillDAG (5.6 points). The oracle averaged four more completed tasks, an 11.1-point gap.

5 Conclusion

SkillWorld combines recompilable skill graphs, read-only applicability projection, and source-authorized ontology updates. Evolved static graphs improve success over base graphs and tested baselines on 72 SkillsBench tasks for both models. ALFWorld ablations support current-state retrieval and joint contributions from relation annotations and candidate organization. SkillWorld achieves the highest non-oracle mean success on curated difficult internal-service tasks.

AI USE STATEMENT

We used generative AI to assist with research software implementation and debugging, literature discovery, manuscript drafting and editing, and mathematical proof development and formalization in Lean 4. The authors reviewed assisted code, mathematics, proofs, citations, and empirical interpretations against the corresponding artifacts and sources and take responsibility for all final content.

REPRODUCIBILITY STATEMENT

Section 4 details the evaluation protocols, while Appendices C and D state the proof assumptions and trusted boundary. The supplementary code archive contains the core SKILLWORLD implementation, Lean 4 proofs and native checkers, the SkillsBench Qwen and fixed ALFWorld ontology snapshots with their corresponding skill manifests, task-execution interfaces for both benchmarks, and build and audit instructions.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022. arXiv:2204.01691.

Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 68–79, 1999. doi: 10.1145/303976.303983.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007. doi: 10.1017/CBO9780511711787.

Rahul Suresh Babu and Laxmipriya Ganesh Iyer. ToolChoiceConfusion: Causal minimal tool filtering for reliable LLM agents. *arXiv preprint arXiv:2606.06284*, 2026. doi: 10.48550/arXiv.2606.06284.

Akhil Bagaria, Jason K. Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 521–531. PMLR, 2021. URL `https://proceedings.mlr.press/v139/bagaria21a.html`.

Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W. Tsang. SkillDAG: Self-evolving typed skill graphs for LLM skill selection at scale. *arXiv preprint arXiv:2606.03056*, 2026. doi: 10.48550/arXiv.2606.03056.

Debangshu Banerjee, Changming Xu, Eugene Ie, Ming Zhang, Daiyi Peng, Chu-Cheng Lin, and Gagandeep Singh. SEVerA: Verified self-evolving agents. *arXiv preprint arXiv:2603.25111*, 2026. doi: 10.48550/arXiv.2603.25111. URL `https://arxiv.org/abs/2603.25111v2`.

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):281–300, 1997. doi: 10.1016/S0004-3702(96)00047-1.

Wenjie Chen, Wenbin Li, Di Yao, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph enhanced tool planning with large language model. *arXiv preprint arXiv:2508.12725*, 2025. doi: 10.48550/arXiv.2508.12725.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

Yu Feng, Ruben Martins, Yuepeng Wang, Isil Dillig, and Thomas W. Reps. Component-based synthesis for complex APIs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pp. 599–612, 2017. doi: 10.1145/3009837.3009851.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5.

Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: classification and survey. *The Knowledge Engineering Review*, 23(2):117–152, 2008. doi: 10.1017/S0269888908001367.

Dawei Fu, Cheng Jiang, Sitian Qian, Huainan Wang, and Zhongkai Hao. SE-GoS: Self-evolving graph-of-skills for skill library at scale. *arXiv preprint arXiv:2609.08228*, 2026. doi: 10.48550/arXiv.2609.08228.

Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2007. doi: 10.1145/1265530.1265535.

Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 157–166, 1993. doi: 10.1145/170035.170066.

Yu Hao, Jinxuan Cai, Qi Zhang, Yawen Li, Zhiqiang Zhang, Chuan Shi, and Cheng Yang. HiSkill: Empowering LLM agents with hierarchical skill graphs. *arXiv preprint arXiv:2607.25853*, 2026. doi: 10.48550/arXiv.2607.25853.

Ken Huang and Jerry Huang. Audited skill-graph self-improvement for agentic LLMs via verifiable rewards, experience synthesis, and continual memory. *arXiv preprint arXiv:2512.23760*, 2025. doi: 10.48550/arXiv.2512.23760.

Herbert Jordan, Bernhard Scholz, and Pavle Subotić. Soufflé: On synthesis of program analyzers. In *Computer Aided Verification (CAV)*, pp. 422–430, 2016. doi: 10.1007/978-3-319-41540-6_23.

Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. doi: 10.1613/jair.5575.

Michael Leuschel and Maurice Bruynooghe. Logic program specialisation through partial deduction: Control issues. *Theory and Practice of Logic Programming*, 2(4–5):461–515, 2002. doi: 10.1017/S147106840200145X.

Xiangyi Li et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026a. doi: 10.48550/arXiv.2602.12670.

Xiaoyuan Li, Moxin Li, Keqin Bao, Yubo Ma, Wenjie Wang, Dayiheng Liu, and Fuli Feng. SkillGraph: Skill-augmented reinforcement learning for agents via evolving skill graphs. *arXiv preprint arXiv:2605.12039*, 2026b. doi: 10.48550/arXiv.2605.12039.

Yubo Li. Dynamic agent skills: A lifecycle survey and taxonomy of evolving skill libraries. *arXiv preprint arXiv:2607.10113*, 2026. doi: 10.48550/arXiv.2607.10113.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. arXiv:2209.07753.

Yuan Liang et al. SkillNet: Create, evaluate, and connect AI skills. *arXiv preprint arXiv:2603.04448*, 2026. doi: 10.48550/arXiv.2603.04448.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026a. doi: 10.48550/arXiv.2604.05333.

Hao Liu and Dongyu Li. SkillGraph: Graph foundation priors for LLM agent tool sequence recommendation. *arXiv preprint arXiv:2604.19793*, 2026. doi: 10.48550/arXiv.2604.19793.

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*, 2024. doi: 10.48550/arXiv.2403.00839.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, Xiyu Ren, Chenyu Ruan, Haoran Li, and Yangqiu Song. SkillRevise: Improving LLM-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026b. doi: 10.48550/arXiv.2606.01139.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. ControlLLM: Augment language models with tools by searching on graphs. *arXiv preprint arXiv:2310.17796*, 2023. doi: 10.48550/arXiv.2310.17796.

David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic markup for web services. W3C member submission, World Wide Web Consortium, 2004. URL `https://www.w3.org/Submission/OWL-S/`.

Natalya F. Noy and Michel Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4):428–440, 2004. doi: 10.1007/s10115-003-0137-2.

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 *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.16789.

Luciano Serafini and Martin Homola. Contextualized knowledge repositories for the semantic web. *Journal of Web Semantics*, 12–13:64–87, 2012. doi: 10.1016/j.websem.2011.12.003.

Zayx Shawn. PACE: Anytime-valid acceptance tests for self-evolving agents. *arXiv preprint arXiv:2606.08106*, 2026. doi: 10.48550/arXiv.2606.08106.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. SkillFoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026. doi: 10.48550/arXiv.2604.03964.

Haochen Shi, Xingdi Yuan, and Bang Liu. Evolving programmatic skill networks. *arXiv preprint arXiv:2601.03509*, 2026. doi: 10.48550/arXiv.2601.03509.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.

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 (ICLR)*, 2021. arXiv:2010.03768.

Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004. doi: 10.1016/j.websem.2004.06.005.

Xinyuan Song, Hongji Pu, and Liang Zhao. SkillOps: Managing LLM agent skill libraries as self-maintaining software ecosystems. *arXiv preprint arXiv:2605.13716*, 2026. doi: 10.48550/arXiv.2605.13716.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

Bo Tang, Yang Zhang, Guomian Zhuang, Wenqiang Wei, Gaoyang Zheng, Lindong Xie, Yanchao Tan, Feiyu Xiong, Qingyu Yang, Edward Chung, and Zhiyu Li. From memory to skills: Evidence-grounded co-evolution governance for long-horizon LLM agents. *arXiv preprint arXiv:2607.16621*, 2026. doi: 10.48550/arXiv.2607.16621.

Chenwei Tang, Lin Long, Xinyu Liu, Jingyu Xing, Zizhou Wang, Joey Tianyi Zhou, Jiawei Du, Liangli Zhen, and Jiancheng Lv. SAG-Agent: Enabling long-horizon reasoning in strategy games via dynamic knowledge graphs. *arXiv preprint arXiv:2510.15259*, 2025. doi: 10.48550/arXiv.2510.15259.

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.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2409.07429.

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. doi: 10.48550/arXiv.2604.17870.

Yue Yao, Shengyuan Wang, Xin Chen, Minke Zhang, Jia He, Bingjun Luo, and Tom Gedeon. SkillTrace: Traversing a query-skill graph for composable LLM agents. *arXiv preprint arXiv:2608.02356*, 2026. doi: 10.48550/arXiv.2608.02356.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.17428.

Zian Zhai, Xingyu Tan, Gaowang Zou, Xiaoyang Wang, and Wenjie Zhang. HyperAgent: Planning and acting over tool–schema hypergraphs for tool-use LLM agents. *arXiv preprint arXiv:2608.02650*, 2026. doi: 10.48550/arXiv.2608.02650.

Di Zhang. AgentDevel: Reframing self-evolving LLM agents as release engineering. *arXiv preprint arXiv:2601.04620*, 2026. doi: 10.48550/arXiv.2601.04620.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. CoEvoSkills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026a. doi: 10.48550/arXiv.2604.01687.

Wentao Zhang, Zhe Zhao, Haibin Wen, Yingcheng Wu, Cankun Guo, Ming Yin, and Bo An. Autogenesis: A self-evolving agent protocol. *arXiv preprint arXiv:2604.15034*, 2026b. doi: 10.48550/arXiv.2604.15034.

Xinyu Zhao, Zhen Tan, Vaishnav Tadiparthi, Nakul Agarwal, Kwonjoon Lee, Ehsan Moradi Pari, Hossein Nourkhiz Mahjoub, and Tianlong Chen. Generative skill composition for LLM agents. *arXiv preprint arXiv:2606.32025*, 2026. doi: 10.48550/arXiv.2606.32025.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

APPENDIX

A ASSUMPTIONS AND TRUSTED BOUNDARIES

Projection assumes typed manifests, finite symbolic states, sound grounding and goal candidates, deterministic hard effects, and a finite release candidate graph. Unobserved state yields unknown edges. The accuracy of manifest extraction and grounding determines the reliability of decided judgments; stochastic or long-horizon effects require richer models.

The source-consistency checker establishes exact support and semantic agreement over normalized, frozen inputs. Source-bound publication additionally assumes successful authorization for the active parent, exact decoding of the materialized successor, and bundle-checker acceptance. Action-language realizability assumes supplied coordinates and target values. Refresh budgeting assumes complete call accounting and adapter enforcement of reserved input and output limits. Grounding, source-review judgments, and the environmental semantics of skill descriptions supply the semantic inputs to the analysis.

Implementation obligations cover source capture and extraction, faithful normalization, canonical JSON projection, hashing, correspondence between decoded artifacts and the semantic store, and atomic host publication. Source groups record declared provenance. Appendix D specifies the checks and trusted computing base.

B FULL RELATION SEMANTICS

This appendix gives the discovery roles, offline witness conditions, and online activation rules for the five relations introduced in Section 3.1.2.

B.1 CANDIDATE DISCOVERY AND STRUCTURED CONTEXT

Table 4 summarizes retrieval from either endpoint and current activation for each relation.

Table 4: Discovery and activation for a potential relation $a\,r\,b$. The first four relations expand candidates; conflicts supply constraints. All rows are subject to administrative quarantine.

Relation	Discovery from either endpoint	Current activation
`CanFeed`	At b: provider a. At a: consumer b.	Both applicable; matched ports bind to the same available resource of the expected kind.
`Enables`	At b: supporter a. At a: continuation b.	a applicable, b blocked; $T_a(z_t)$ is defined and makes every hard precondition of b hold.
`AlternativeTo`	Either endpoint retrieves the other as an alternative candidate.	Both applicable and in the current goal's candidate set.
`Specializes`	At b: specialized variant a. At a: general variant b.	Semantically true in every context; endpoint applicability is evaluated separately.
`ConflictsWith`	Either endpoint retrieves conflict context for existing candidates.	Both applicable; in at least one direction, executing one falsifies a hard precondition of the other.

Candidate discovery. A retriever supplies query-relevant seed skills. SKILLWORLD performs bounded traversal of the release's potential `CanFeed`, `Enables`, `AlternativeTo`, and `Specializes` relations from either endpoint. Traversal preserves the original edge and its direction. The candidate and hop budgets apply equally to confirmed, unknown, and semantically rejected relations; quarantined edges and endpoints are excluded. Retrieval scores and seed rank resolve truncation when the candidate budget is exceeded. Conflict edges touching the resulting pool supply constraint context without adding recommendations. Within the hop budget, traversal can recursively collect partial enabling support.

Structured context. The context groups records by candidate skill and relation type. Each skill record contains its identifier, summary, and declared interface: inputs, outputs, preconditions, and effects. Each relation record preserves its endpoints, type, direction, and supporting witness; discovery provenance identifies the seed and path. When runtime context is available, projection adds current skill applicability, relation status, and typed reasons under the observed state and resource bindings. Skill applicability and relation status are separate fields. Static retrieval retains declared conditions without asserting current applicability. Candidate and relation budgets bound the context, with omitted relation evidence marked as truncated.

Model selection. The language model receives the task and these records, and selects skills within the document budget by considering task relevance, complementary roles, and current conditions. Its output identifies skills from the candidate pool. The model uses these state and relation records to determine candidate ranking and the treatment of alternatives and conflicts. Unknown labels communicate uncertainty; semantic rejection marks a relation as currently inactive while preserving its structural evidence for planning. A blocked `RunTests` candidate can therefore be selected together with `PrepareEnv` to address its missing dependency. Execution requires the selected skill's hard preconditions to hold. The selected documents are loaded with their relation context, and subsequent calls refresh state annotations using updated observations and bindings.

B.2 Offline Witnesses

Let $\mathrm{cap}(s) = (n_s, o_s, t_s)$ be the primary namespace, operation class, and target of skill s. Every clause below requires $s_i \neq s_j$ and valid primary capabilities; all named modes and gates are read from the release-pinned RBox. Table 5 states the normalized conditions used by the offline compiler:

Table 5: Exact offline relation schemas. *Gate* denotes the configured relation-specific exclusion predicate, and $\mathrm{reqCompat}$ is true when the skills share a required input kind or both have no required inputs.

Relation	Compiler condition for $s_i \to s_j$
`Specializes`	Exactly one capability axis follows a direct configured cover while the other two are equal: $t_i \prec t_j$, $o_i \prec o_j$, or $n_i \prec n_j$. The involved targets are non-abstract.
`CanFeed`	s_i has producer role and an output port of kind a; s_j has an input port of kind b; $\mathrm{resourceMatch}_m(a, b)$ holds for configured mode $m \in \{\textsf{exact}, \textsf{from-isa-to}\}$; the exact match is not a configured generic-target match; and $\neg\,\mathrm{Gate}(s_i, s_j)$.
`Enables`	A canonical effect of s_i and canonical precondition of s_j have the same subject target, state dimension, and valid value.
`AlternativeTo`	The capabilities are either identical or have the same namespace and operation with sibling targets, as enabled by the configured mode. The targets are not configured exclusions; identical targets are non-abstract; $\mathrm{reqCompat}(s_i, s_j)$ and $\neg\,\mathrm{Gate}(s_i, s_j)$ hold. The scope witness is a shared domain for general pairs, or the same nonempty declared hard-effect contract for pairs subject to the typed scope gate. Finally, $s_i < s_j$ canonicalizes the symmetric pair.
`ConflictsWith`	One configured contradiction witness holds: an effect of s_i contradicts a precondition of s_j on the same subject target and dimension; effects of s_i and s_j contradict on the same subject target and dimension (stored with $s_i < s_j$); or destructive s_i opposes a required `exists=true` precondition of s_j.

Each derivation stores the matched capabilities, ports or conditions, RBox axiom, projection mode, and source fact identifiers. Thus the table specifies candidate generation with an explicit record of each derivation.

B.3 Runtime Fact Interface

Applicability and transitions. Activation is built from applicability and declared transitions. Let γ_{b_t} replace each role subject by its single explicitly bound entity in b_t. All conditions on that role and target type share this entity; a missing or multi-candidate binding leaves those conditions unresolved. Observed type membership alone does not bind a role. A skill's applicability is the Strong–Kleene

conjunction of its grounded hard preconditions,

$$A_s(z_t) = \bigwedge\nolimits_{\mathbb{K}_3} \mathrm{lookup}_{x_t}(\gamma_{b_t}(P_s^{\mathrm{hard}})),$$

and $T_s(z_t)$ applies a deterministic $\gamma_{b_t}(E_s^{\mathrm{hard}})$. Unresolved atoms yield U; ambiguous hard effects are rejected at publication. Each formula reads only the atoms relevant to its candidate relation. Partial inputs can yield a decided judgment: a false conjunct makes a conjunction false, and a true disjunct makes a disjunction true. Unresolved cases remain unknown. A confirmed edge supports its current relation-specific use; an unknown or semantically rejected edge retains its structural witness for discovery.

These semantics assume grounding preserves output provenance, supplies sound goal candidates, and marks directional conflict only when a post-state violates a peer hard precondition (Appendix A).

Relation facts. The finite EdgeFacts interface used by Table 6 has the following meanings: *source/target_applicable* are $A_{s_i}(z)$ and $A_{s_j}(z)$; *target_after_source* states that every target precondition holds after $T_{s_i}(z)$; the two *goal_candidate* atoms encode current-goal membership; *feed_witness* records equal compatible port bindings to an available resource; and each directional *conflict* records that an applicable source's post-state falsifies a peer precondition. Missing input is U; $q_k(e)$ is a distinct administrative override.

B.4 Online Activation

Table 6: Semantic activation formulas before the shared release quarantine override; T, U, and F map to confirmed, unknown, and rejected.

Relation	Verified Strong–Kleene formula
Specializes	T
Enables	$(source_applicable \wedge (\neg target_applicable \wedge target_after_source))$
ConflictsWith	$(forward_conflict \vee reverse_conflict)$
CanFeed	$(source_applicable \wedge (target_applicable \wedge feed_witness))$
AlternativeTo	$(source_goal_candidate \wedge (target_goal_candidate \wedge (source_applicable \wedge target_applicable)))$

`Enables`. The offline witness matches one effect to one precondition. Current activation requires an applicable source a, a blocked target b, and a defined transition $T_a(z_t)$ after which every hard precondition of b holds. A blocked source or partial support can leave the edge rejected while its structural witness remains useful for candidate discovery.

`ConflictsWith`. Each displayed edge denotes a symmetric candidate pair. For each evidence direction $s_i \to s_j$, the direction is confirmed when both skills are currently applicable and executing s_i makes at least one hard precondition of s_j false. The pair is active if at least one direction is confirmed, and projection retains every confirmed direction as a witness. An offline effect–effect contradiction creates a potential pair; runtime activation additionally requires a witnessed violation of a peer hard precondition.

`CanFeed`. Typed port compatibility is one of several activation requirements. Both skills must be applicable; the matched output and input ports must bind to the same resource identifier; and the current state must mark that resource as available with the expected kind. Missing bindings can leave activation unknown; unavailable resources or kind mismatches can reject it. Projection reports the corresponding diagnostic reasons.

`AlternativeTo`. Both skills must be members of the current goal's candidate set and currently applicable. This prevents a library-wide similarity relation from being treated as a choice when only one skill addresses the current goal or can execute in the current state.

`Specializes`. Taxonomic specialization has semantic value T in every episode and is therefore state-independent. The release quarantine override remains applicable.

C Projection Proof and Snapshot-Store Contract

Local completeness. Completeness is relation-local. For a candidate edge e, let

$$\mathcal{D}_e = \mathrm{Deps}(\phi_{r(e)})$$

be the summary atoms read by its formula. Context z is *locally complete* for e when every atom in the finite set $\mathcal{D}_e$ is decided. A completion $\widehat{z} \succeq_e z$ decides $\mathcal{D}_e$ while preserving decided atoms and quarantine.

We mechanize the projection contract in Lean 4 (de Moura & Ullrich, 2021). The supplementary code archive includes the projection implementation and accompanying Lean 4 proofs.

Proposition 1 (Local-completeness exactness and completion soundness)**.** *For every potential edge e and context z:*

1. *if z is locally complete for e and e is not quarantined, then e is not unknown, and its confirmed and rejected partitions coincide exactly with $\phi_{r(e)}(e, z) = \mathsf{T}$ and $\phi_{r(e)}(e, z) = \mathsf{F}$, respectively; and*
2. *for every completion $\widehat{z} \succeq_e z$, a confirmed judgment at z remains true at $\widehat{z}$, and a non-quarantine rejected judgment at z remains false at $\widehat{z}$.*

Proof of Proposition 1. Lean defines the information order by $\mathsf{U} \preceq \mathsf{F}$, $\mathsf{U} \preceq \mathsf{T}$, and $v \preceq v$. Structural induction on the finite formula IR proves two facts. First, if every atom in the mechanically extracted dependency set is decided and the formula contains no unknown constant, its evaluation is not U. Second, Strong–Kleene negation, conjunction, and disjunction are monotone in this information order. Thus a decided formula value equals its value in every completion that preserves decided atoms. Instantiating these lemmas with each generated relation formula and unfolding the partition function gives `local_complete_exact`, `confirmed_sound_under_completion`, and `rejected_sound_under_completion`. The quarantine side condition restricts the rejection guarantee to judgments supported by a false activation predicate. Quarantine records a separate administrative rejection. □

The formal projection also has total, pairwise-disjoint annotations, preserves edge origin, leaves the release unchanged, and supplies a reason for every rejection. Runtime termination, canonical determinism, and the absence of store writes are checked by implementation tests.

Snapshot isolation and release pinning (S0). Assume that an acquired bundle's bytes and decoded objects remain unchanged throughout a projection call, and publication atomically updates the pointer used by later acquisitions. Concurrent publication then affects only later calls. Each call returns its release identifier; clients requiring consistency across an episode reject identifier changes. The snapshot contract relies on filesystem atomicity as a separate implementation assumption, assessed by tests.

D Self-Governance: Authorization and Publication Contracts

This appendix formalizes source authorization and its connection to model updates. The finite input contains the frozen candidate menu, evidence pool, source atoms reparsed from frozen manifests, and accepted retractions. A native Lean checker decides the authorization predicate used by the runtime. The publication contract connects an authorized edit to a validated successor through exact materialization and the bundle checker.

Candidate normalization and selection. For namespace, operation-class, target, and state records, grouping compares all fields except the top-level `description`. Each candidate retains one complete source value, including its description, for exact checking. Description-only differences from the current record are treated as equivalent; distinct semantic values remain separate candidates with explicit conflict references. The selector returns one existing `choice_id` or `abstain`. An invalid response permits one retry, followed by abstention if the retry is also invalid. Creation, update, and deletion share canonical-JSON $\mathsf{Put}(c, v)$: prior absence distinguishes creation from update, and $v = \texttt{None}$ denotes deletion.

Source-consistency check and authorization. Let F contain a parent release, a frozen candidate menu, an evidence pool, source atoms reparsed from frozen manifests, and verifier-accepted retractions. A candidate q binds an identifier, parent, coordinate c, and complete value v. Each atom retains its complete canonical record and a semantic projection used only for agreement checks. For hierarchical records, that projection omits the top-level `description`; exact support still compares the complete value.

The predicate $\mathsf{SourceValid}(F, q)$ requires a nonempty parent and candidate identifier, membership in the frozen menu, and a matching parent. It further requires a live artifact atom present identically in both the pool and the reparsed source atoms, with exactly the proposed coordinate and complete value. Every live atom for that coordinate in either collection must agree under the semantic projection. `SourceBoundEvidence.lean` defines this predicate independently of its finite Boolean checker and proves

$$\mathsf{checkSource}(F, q) = \mathsf{true} \iff \mathsf{SourceValid}(F, q).$$

The model supplies only a choice identifier. Issuance looks it up in the frozen menu and returns a token only when the checker accepts. Consumption rechecks validity and exact binding to the frozen context, parent, coordinate, and complete value.

Theorem 2 (Source-bound write authorization)**.** *Every issued token identifies a candidate in the frozen menu with live, exact source support. An empty source-atom collection cannot authorize any candidate, and a live semantic conflict in either the evidence pool or reparsed source atoms prevents issuance. A consumed token authorizes only its bound parent, context, coordinate, and complete value. Changing the write payload, including its description, cannot transfer that authorization to the new value.*

The checker equivalence is `check_iff`. Theorems `issued_token_authorizes`, `accepted_exact_source`, `no_replay_no_authorization`, and `conflict_blocks` establish issuance and source support. Theorem `consume_iff` connects the consumption checker to exact authorization; `authorization_not_transferable`, `stale_parent_rejected`, and `changed_source_rejected` exclude substitution.

Action realization. Create, replace, and delete use the existing canonical-JSON $\mathsf{Put}(c, v)$ semantics. A single authorized write assigns v at c and leaves every other coordinate in the mutable ontology store unchanged. More generally, for a finite, duplicate-free coordinate list W and a supplied target $S^\star$ that agrees with S_0 outside W, assigning each target value yields

$$\mathsf{applyActions}(S_0, \mathsf{compileActions}(S^\star)) = S^\star.$$

`SelfEvolutionCompleteness.lean` proves this action-language result as `compiled_actions_reach_target`. This result establishes the action language's expressiveness for supplied targets satisfying the stated finite-scope conditions.

Connection to publication. Let $\Sigma_k = \pi_{\mathrm{edit}}(\mathcal{R}_k)$ denote the coordinate-indexed store of mutable ontology assertions, including explicit assertion overrides. Extracted manifests, compiled relations, and release metadata are separate components of the release. `successorOfIssued` constructs a successor from successful source issuance, a matching active parent, a distinct successor identifier, and an accepted materialized bundle. Exact materialization requires

$$\pi_{\mathrm{edit}}(\mathcal{R}_{k+1}) = \mathsf{Put}(\Sigma_k, c, v).$$

The frame property keeps its other ontology coordinates unchanged. Before publication, Algorithm 1 prepares a reviewed manifest collection M^+ under the staged ontology Y_{k+1}. The materialized bundle contains M^+ and $G^\star_{k+1} = \mathcal{C}(M^+, Y_{k+1})$ and must pass $\mathsf{checkWF}$. The refresh theorem below establishes review coverage for M^+.

Theorem 3 (Source-bound publication invariants)**.** *Let $\mathcal{R}_k$ be an active well-formed release. Suppose source issuance succeeds for a candidate bound to that parent, the successor's mutable ontology store equals the selected write applied to Σ_k, its release identifier differs from the parent's, and* $\mathsf{checkWF}$ *accepts it. A successful CAS activates that exact successor, increments the generation once, and preserves well-formedness. Every changed coordinate in this store and its complete final value have a live artifact atom present identically in the frozen pool and reparsed source atoms. All other coordinates in this store remain unchanged. A stale or failed CAS leaves the complete active snapshot unchanged.*

published_bundle_exact, published_change_has_exact_source, publication_preserves_wf, and publication_failure_identity mechanize these properties in a state machine that receives the host CAS outcome as input. Companion implementation tests cover structured-source parsing and source-consistency checks, correspondence between canonical-JSON actions and Lean semantics, hash-tampering detection, atomic publication, and rollback after failure.

Source discovery and selective re-extraction. AFFECTEDSKILLS first selects direct dependencies: source proposals supporting a changed coordinate and manifests already using its vocabulary entry. Skill-specific assertions select their named skill; global state or axiom changes select the full library. Batched edits take the union of these sets. For each vocabulary edit, local TF–IDF retrieval searches the original skill documents using the changed term, its definition, and supporting skill descriptions. Word features and character trigrams connect related expressions without model calls. Cosine scores of at least 0.07 produce a ranked pool; the highest-ranked additional skill enters applicability review, with deterministic identifier tie-breaking. The audit records pool size and truncation.

Applicability review receives the edit, current capability, skill description, and at most 1,200 characters of source excerpts with line numbers. Each candidate requires an applicable, not-applicable, or uncertain decision. Decisions bind to the source hash, current manifest, and exact edit; cited quotes must occur on the stated source lines. Applicable candidates join the direct extraction set; an uncertain or missing decision blocks the round. Each selected skill is re-extracted once from its full original document under the staged ontology, and other manifests retain their original bytes. Re-extraction must preserve skill identity and source hashes.

Semantic review, budgets, and publication. Before compilation, the system compares capabilities, ports, preconditions, effects, constraints, domains, and tools against their previous values. Independent source review must support every semantic change, including added or removed ports, requiredness, target types, and condition or effect subjects. Omitted port requiredness means true; annotation-only changes do not require review. Unsupported changes, uncertain decisions, missing evidence, and changed source hashes block publication. Source quotations bind each review to its cited evidence.

All applicability, extraction, and semantic-review calls share a round token budget; a separate cap bounds full-document re-extractions. Reviews use batches of at most eight items. Before dispatch, the adapter reserves an input-token upper bound plus the output limit, which defaults to 4,096 tokens. Reported usage replaces the reservation; missing usage retains the reservation and blocks further calls. Exceeding a budget is recorded as a blocked round. CHECKEDINPUT compiles the reviewed manifests and checks vocabulary, library membership, graph consistency, and configured task cases. A receipt binds the review and validation records to the exact rules, manifests, and graph hashes. Only this checked bundle proceeds to checkWF and parent-bound atomic publication; the next round reads the activated bundle.

Checked refresh contract. CheckedRefresh.lean represents a refresh by finite normalized manifests, source lines, review responses, artifact hashes, and model-call records. Its executable checker computes selection and changed semantic fields, then checks review coverage, extraction counts, token accounting, and artifact bindings.

Theorem 4 (Checked refresh safety)**.** *For every refresh accepted by this checker:*

1. *every selected skill appears exactly once in the extraction trace; unselected manifests retain their bytes, and source identities are preserved;*
2. *every changed normalized semantic field has a supporting review bound to the source, staged ontology, and before/after manifests, with matching source quotations;*
3. *recorded token charges and extraction counts respect their respective limits; missing usage or an exhausted reservation budget prevents acceptance.*

Composing these gates with source-authorized publication preserves bundle well-formedness and exact write authorization. Rejected refreshes and stale or failed CAS outcomes leave the complete active snapshot unchanged. Successful publication activates the exact bundle bound to the reviewed artifact hashes.

Proof. `changed_field_discovered` covers additions, deletions, and value changes. `accepted_change_reviewed`, `accepted_frame`, and `accepted_extracted_once` establish the manifest obligations. Induction over the call trace proves `ready_within_budget`. `activated_requires_checked_refresh` binds admission to publication; `publication_preserves_wf` and `published_change_has_exact_source` compose with the preceding source-bound theorems. □

Python–Lean correspondence tests check concrete budget traces and semantic differences.

Projection implementation. `ProjectionIR.lean` defines the finite Strong–Kleene formula IR and lowers it to a relational abstract syntax tree. For every relation r in Table 6 and finite environment ρ, it proves

$$[\![\mathsf{lower}(\phi_r)]\!]_\rho = [\![\phi_r]\!]_\rho = \mathsf{semanticTruth}(r, \rho).$$

Evaluation reads only mechanically extracted dependencies, so changing other grounded atoms leaves edge truth unchanged. Appendix C states the local-completeness and completion guarantees. A shared emitter produces the relation formulas and reason codes for Datalog, Python, and this paper. The lowering proof establishes semantic preservation from the formula IR to the target AST. Schema checks and differential tests assess correspondence with executable Datalog.

Publication checker. `ReleaseBundle.lean` defines an executable checker $\mathsf{checkWF} : \mathsf{Bundle} \to \mathsf{Bool}$ and an independently specified predicate $\mathsf{WF} : \mathsf{Bundle} \to \mathsf{Prop}$, and proves

$$\mathsf{checkWF}(B) = \mathsf{true} \iff \mathsf{WF}(B).$$

The predicate covers schema and parent identities, artifact hashes, canonical ordering and reference closure, relation-specific edge shapes, hard conditions, deterministic effects, and typed primitive values. The native checker runs before publication. Parsing and fields outside the checked representation remain separate implementation obligations.

Native source-consistency checker. `SourceBoundEvidenceMain.lean` decodes canonical coordinate/value records and invokes the proved functions for identifier selection and source-consistency checking. The Python path reconstructs the candidate from the frozen pool, locates its supporting manifests by skill identifier, and deterministically parses the structured proposals in their `unknowns` fields. The resulting atoms retain the source field, supporting text, coordinate, and complete proposed value. The native checker runs at both certificate issuance and consumption, and both stages require acceptance. The certificate binds the parent release, candidate, exact coordinate and complete value, and hashes of the evidence pool, newly parsed source atoms, and accepted retractions; an exact write token binds the authorized edit. At consumption, the system reparses the frozen manifests, regenerates the certificate and token, and compares them exactly with the supplied records. It also checks that the edit payload matches the authorized coordinate and complete value. For structured manifest proposals, the automatic source-consistency check covers namespaces, operation classes, targets, and states. State proposals retain their complete records, including hierarchy, domain, and value-space fields. Tests exercise missing sources, altered atoms and payloads, hidden conflicts, invalid choices, and stale parents, including mandatory checker acceptance at both runtime stages. Statement and axiom audits accompany the Lean build.

Publication and rollback. Projection reads $\mathcal{R}_k$ while repair constructs $\mathcal{R}_{k+1}$. After plan validation and bundle-checker acceptance, a compare-and-swap (CAS) activates the successor when its parent is still current. A failed check or stale parent preserves the active release. Rollback applies the same freshness check to restore an accepted, healthy ancestor.

Implementation checks and trusted computing base. The trusted computing base comprises Lean's kernel and native runtime, parsing, hashing, the filesystem, pinned Soufflé and generated C++, and handwritten grounding into the finite IR. Frozen-source capture and extraction, semantic-field comparison, exact materialization, and atomic publication are implementation obligations. The supplement checks generated schemas, the five relation implementations, and JSON Put actions against their formal interfaces, and tests publication with valid, tampered, and stale bundles.

E SkillsBench Evaluation Task Set

The evaluation in Section 4.1 uses a fixed 72-task evaluation subset of 87 SkillsBench tasks. A task is excluded only if all three attempts at that task fail with infrastructure errors; this criterion accounts for the other 15 tasks. Ordinary task or verifier failures do not qualify a task for exclusion. Table 7 lists all 72 retained task identifiers. Every model–method configuration uses this same task set for three runs. Within each run, unsuccessful attempts, including any infrastructure errors, remain in the 72-task success-rate denominator. Reported passed counts and success rates are averaged across the three runs.

Table 7: The fixed 72-task evaluation subset used in the SkillsBench experiments. Identifiers are ordered lexicographically down the left column, then the right.

No.	Task ID	No.	Task ID
1	3d-scan-calc	37	lake-warming-attribution
2	ada-bathroom-plan-repair	38	lean4-proof
3	adaptive-cruise-control	39	llm-prefix-cache-replay
4	azure-bgp-oscillation-route-leak	40	manufacturing-codebook-normalization
5	bike-rebalance	41	manufacturing-equipment-maintenance
6	citation-check	42	manufacturing-fjsp-optimization
7	civ6-adjacency-optimizer	43	mario-coin-counting
8	court-form-filling	44	mars-clouds-clustering
9	crystallographic-wyckoff-position-analysis	45	offer-letter-generator
10	dapt-intrusion-detection	46	organize-messy-files
11	data-to-d3	47	paper-anonymizer
12	dialogue-parser	48	paratransit-routing
13	drone-planning-control	49	pddl-airport-planning
14	dynamic-object-aware-egomotion	50	pddl-tpp-planning
15	earthquake-plate-calculation	51	pdf-excel-diff
16	econ-detrending-correlation	52	powerlifting-coef-calc
17	edit-pdf	53	pptx-reference-formatting
18	energy-ac-optimal-power-flow	54	protein-expression-analysis
19	energy-market-pricing	55	quantum-numerical-simulation
20	energy-unit-commitment	56	r2r-mpc-control
21	enterprise-information-search	57	radar-vital-signs
22	exam-block-sequencing	58	react-performance-debugging
23	exceltable-in-ppt	59	reserves-at-risk-calc
24	exoplanet-detection-period	60	sales-pivot-analysis
25	financial-modeling-qa	61	sec-financial-report
26	fix-erlang-ssh-cve	62	setup-fuzzing-py
27	fix-visual-stability	63	shock-analysis-demand
28	flood-risk-analysis	64	shock-analysis-supply
29	glm-lake-mendota	65	spring-boot-jakarta-migration
30	gravitational-wave-detection	66	suricata-custom-exfil
31	grid-dispatch-operator	67	threejs-structure-parser
32	hvac-control	68	threejs-to-obj
33	invoice-fraud-detection	69	tictoc-unnecessary-abort-detection
34	jax-computing-basics	70	video-silence-remover
35	jpg-ocr-stat	71	weighted-gdp-calc
36	lab-unit-harmonization	72	xlsx-recover-data

F ALFWorld Results Across Repeats

Table 8 reports the three repeats of the same 134 unseen games under the protocol in Section 4.2. Each repeat contains 134 episodes; the overall rate pools all 402 episodes. For both models, Full has higher SR@15 than No relation labels, Static-default, and Static-random in each repeat. No relation labels retains dynamic state and individual skill applicability, with Full's candidate-pool construction, models, and budgets. The executor orders candidates by individual applicability and their original pool order, and presents typed relations without projection labels. Full additionally uses confirmed relations to promote supported candidates, defer alternatives, and select relation context.

SkillDAG evolves from execution trajectories, whereas SkillWorld evolves its ontology from skill artifacts and compilation diagnostics before execution. The SkillDAG comparison evaluates complete systems; No relation labels evaluates the combined contribution of relation annotations and candidate organization. We report the observed differences in task success rates and invalid-action counts between SkillWorld and the system baselines.

Table 8: ALFWorld SR@15 (%) with successful-episode counts in parentheses. Each repeat has 134 episodes; the overall denominator is 402. Bold marks the highest overall success rate within each model. For GoS with Qwen, invalid-action counts are 103, 105, and 108 across the three repeats (316 overall).

Condition	Repeat 1	Repeat 2	Repeat 3	Overall
Qwen3.8-27B				
Full	85.07 (114)	85.07 (114)	85.07 (114)	**85.07** (342)
No relation labels	75.37 (101)	74.63 (100)	75.37 (101)	75.12 (302)
Static-default	82.84 (111)	81.34 (109)	82.09 (110)	82.09 (330)
Static-random	82.84 (111)	81.34 (109)	78.36 (105)	80.85 (325)
SkillDAG	80.60 (108)	80.60 (108)	82.09 (110)	81.09 (326)
Graph of Skills (GoS)	68.66 (92)	70.15 (94)	70.90 (95)	69.90 (281)
No graph (BM25)	75.37 (101)	75.37 (101)	75.37 (101)	75.37 (303)
No skills	70.15 (94)	70.15 (94)	69.40 (93)	69.90 (281)
DeepSeek V4.1 Flash				
Full	85.07 (114)	86.57 (116)	85.82 (115)	**85.82** (345)
No relation labels	82.84 (111)	76.12 (102)	80.60 (108)	79.85 (321)
Static-default	82.84 (111)	83.58 (112)	80.60 (108)	82.34 (331)
Static-random	80.60 (108)	76.87 (103)	79.85 (107)	79.10 (318)
SkillDAG	85.82 (115)	85.07 (114)	82.09 (110)	84.33 (339)
Graph of Skills (GoS)	73.13 (98)	76.12 (102)	75.37 (101)	74.88 (301)
No graph (BM25)	82.09 (110)	78.36 (105)	81.34 (109)	80.60 (324)
No skills	64.93 (87)	62.69 (84)	65.67 (88)	64.43 (259)

G QWEN EXTRACTION AND TYPE REFINEMENT

This example follows `light-curve-preprocessing` through extraction, an ontology update, re-extraction, and compiled skill relations in the Qwen run of Section 4.1. The recorded model requests specify `qwen3.8-max`.

Skill text and extracted manifest. The skill document describes outlier removal, detrending, flattening, and quality-flag handling before astronomical period analysis. The initial manifest uses `dataset` for its principal input, output, and capability target. Separate `proposed_kind` and `proposed_target` fields suggest `light_curve` as a finer type. Table 9 compares the initial and re-extracted assignments.

Table 9: Selected type assignments in the recorded preprocessing manifests before the ontology update and after re-extraction.

Manifest component	Initial	Re-extracted
Input: `light_curve`	`dataset`	`light_curve`
Output: `preprocessed_light_curve`	`dataset`	`light_curve`
Capability target	`dataset`	`light_curve`

The manifest's first `unknowns` entry proposes `light_curve` under `dataset`, with the description:

> Astronomical time series of flux versus time with associated errors and quality flags, requiring specialized preprocessing such as outlier removal, detrending, and flattening.

Evidence, selection, and exact update. The recorded candidate identifies the target as absent and includes four supporting manifests: `light-curve-preprocessing`, `box-least-squares`, `transit-least-squares`, and `exoplanet-workflows`. They agree on `light_curve` under `dataset`. The selected description quoted above comes from `light-curve-preprocessing.unknowns[0]`.

Qwen returns the identifier of this candidate. The system binds that choice to the recorded coordinate and value and constructs one `Put`:

$$\text{coordinate} = \{\text{kind} : \texttt{target},\ \text{id} : \texttt{light_curve}\},$$
$$\text{value} = \{\text{parent} : \texttt{dataset},\ \text{description} : \text{the quoted source text}\}.$$

The operation is attached to its parent ontology version and evidence reference. The staged TBox contains `light_curve` as a child of `dataset`.

Re-extraction under the updated ontology. The four supporting skills belong to the direct re-extraction set. Source retrieval supplies additional candidates for applicability review as specified in Appendix D. Selected skills are re-extracted from their full original documents under the staged ontology; other manifests are reused. The source hashes remain unchanged. The refreshed preprocessing manifest assigns `light_curve` to its principal input, output, and capability target (Table 9). The proposed-type fields are cleared; its `unknowns` now propose an `astronomy` namespace.

Derived relations and subsequent use. Recompilation adds `CanFeed` edges from `light-curve-preprocessing` to `box-least-squares`, `transit-least-squares`, and `exoplanet-workflows`. For `box-least-squares`, the recorded witness matches the `preprocessed_light_curve` output to the `light_curve` input, both typed `light_curve`. When `box-least-squares` is used as a retrieval seed (Section 3.1.4), this edge makes preprocessing eligible as an upstream candidate and provides an input–output match that the selector can use to assess how the two skills connect. Source review checks the manifest changes before graph validation and publication. The next evolution round reads the published manifests and rebuilt graph.