FROM SKILL TEXT TO SKILL STRUCTURE: THE SCHEDULING-STRUCTURAL-LOGICAL REPRESENTATION FOR AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly rely on reusable *skills*: capability packages that typically combine instructions, control flow, constraints, and calls to external tools and resources. In current agentic systems, however, skills are still represented as text, mainly `SKILL.md`-style documents whose machine-usable evidence remains embedded largely in natural-language descriptions. As a result, skill-centered agentic systems face a representation problem: both managing skill collections and using skills during agent execution require reasoning over invocation interfaces, execution structure, and concrete side effects, but these signals are often entangled in a single textual surface. An explicit representation of skill knowledge may therefore help make these artifacts easier for machines to acquire and leverage. Drawing on Memory Organization Packets, Script Theory, and Conceptual Dependency from Schank and Abelson's classical work on cognitive linguistic representation, we introduce what is, to our knowledge, the first structured representation for agent skill artifacts that disentangles skill-level scheduling signals, scene-level execution structure, and logic-level action/resource-use evidence: the Scheduling-Structural-Logical (SSL) representation. We instantiate SSL with an LLM-based normalizer and evaluate it along two aspects: Skill Routing before a task, and Skill Execution during it. In Skill Routing, indexing candidate skills by their SSL views improves nDCG@10 over the raw view ($+2.69$ pp hard, $+2.95$ pp easy; 6.6% and 6.9% relative); In Skill Execution, an action card that annotates passages of the task's skill files with SSL relations improves verified task success over raw execution ($+5.00$ pp on DeepSeek-V4.1-Flash, $+7.89$ pp on GLM-5.3-Flash; 6.8% and 13.8% relative) at no additional model calls. Together, the two aspects show that an explicit, source-grounded SSL representation helps both when structure selects what the agent sees and when it governs how the agent acts. We position SSL as a practical step toward more inspectable, reusable, and operationally actionable skill representations, and as a building block for future skill registries and agent runtimes.

1 INTRODUCTION

Large language models (LLMs) increasingly drive agentic systems that maintain task context, coordinate multi-step workflows, and act through external resources such as files, tools, or services (Xi et al., 2023; Luo et al., 2025). As these systems move beyond isolated tool calls, reusable capabilities are increasingly packaged as *skills*: bundles of instructions, control flow, constraints, and callable operations that can be discovered, selected, governed, and reused across tasks (Xu et al., 2026; Xu & Yan, 2026; Liang et al., 2026).

Despite this shift, skills are still usually exposed as raw text: `SKILL.md`-style instruction files and README-like documentation, sometimes wrapped in JSON/YAML records whose machine-usable evidence remains in natural-language fields. This reflects a long-standing trade-off in technical documentation: natural language is easy for people to author and read, but difficult for machines to analyze, validate, and reuse reliably (Berners-Lee et al., 2001; González-Mora et al., 2023; Lazar et al., 2025). A single skill artifact can therefore entangle a skill's invocation interface, execution

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

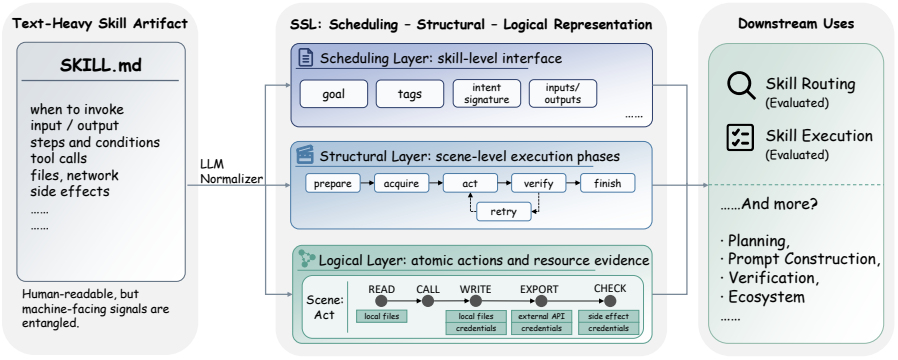

Figure 1: Overview of the SSL representation. A skill artifact is converted by a normalizer into three layers: a scheduling record for skill-level signals, a structural graph of execution scenes, and a logical graph of atomic actions and resource evidence. The structured view remains paired with the original source document and supports downstream uses such as Skill Routing and Skill Execution.

phases, and action/resource-use evidence, forcing downstream components to infer these properties from long, noisy, or potentially incomplete text.

This hurts downstream use of skills: semantically distinct properties are collapsed into a single textual surface. For example, for routing a user task to the most relevant skills, large and overlapping registries require signals beyond sparse metadata like implementation-level cues needed for selection (Zheng et al., 2026; Liu et al., 2026); for implementing skills during execution, success depends on the skill's workflow reaching the agent in a form it can act on, while its execution logic stays buried in instructional prose (Wang et al., 2023; 2024). These observations point to a missing layer between raw skill artifacts and downstream applications: a reusable, source-grounded representation that can expose skill evidence across tasks without repeatedly re-parsing the full instruction text.

To address this gap, we propose the **Scheduling-Structural-Logical (SSL)** representation, which is, to our knowledge, the first structured representation designed specifically for disentangling agent skill artifacts. SSL means to map an unstructured skill document into a typed three-layer JSON graph organized around three analogies from Schank and Abelson's classical work on cognitive linguistic representation: The **Scheduling** layer draws on Memory Organization Packets (MOPs) as goal-oriented organizers for retrieving and contextualizing experience (Schank, 1980); The **Structural** layer draws on Script Theory (ST), which represents stereotyped activities as ordered scenes with expectations and transitions (Schank & Abelson, 1977); The **Logical** layer draws on Conceptual Dependency (CD), which decomposes linguistic meaning into primitive action structures that abstract away from surface wording (Schank, 1972). Together, these theories provide reference framework for disentangling skills into goal-level context, ordered execution trajectory, and primitive operations. An overview of the resulting representation and its role in downstream tasks is illustrated in Figure 1.

While SSL is intended as a general representation for skill-centered agentic systems, we evaluate it in two aspects: Skill Routing, which matches a task to the skills it requires before execution, and Skill Execution, which concerns how the selected skill reaches the agent during the task. For Skill Routing, on the SkillRouter Benchmark (Zheng et al., 2026), it improves nDCG@10 by +2.69 pp on the hard tier and +2.95 pp on the easy tier (6.6% and 6.9% relative), by deriving the recall short lists from the SSL view; for Skill Execution on SkillsBench (Li et al., 2026), an action card that annotates passages of the task's skill files with SSL's relation labels improves verified task success over raw execution ($+5.00$ pp on DeepSeek-V4.1-Flash, $+7.89$ pp on GLM-5.3-Flash; 6.8% and 13.8% relative). Together, the two results show that making skill structure explicit and source-grounded helps both when it selects what the agent sees and when it reaches the agent during execution.

To sum up, we make three contributions in this paper:

- We introduce SSL, a three-layer representation for agent skills that disentangles skill-level scheduling signals, scene-level execution structure, and logic-level action/resource-use evidence, and instantiate it for existing `SKILL.md` files with an LLM-based normalizer;

- We evaluate SSL in two aspects: in Skill Routing, the SSL view improves nDCG@10 over the raw view by +2.69 pp (hard) and +2.95 pp (easy), which is 6.6% and 6.9% relative; in Skill Execution, an action card that annotates passages of the task's skill files with SSL's relation labels improves verified task success over raw executions ($+5.00$ pp on DeepSeek-V4.1-Flash, $+7.89$ pp on GLM-5.3-Flash; 6.8% and 13.8% relative);
- We release the SSL schema and the normalizer prompt, so that researchers can annotate their own skill collections in the same format.

2 Related Work

2.1 Reusable Skills in Agentic Systems

A skill is a capability package that typically combines instructions, control flow, constraints, and calls to external tools and resources. Unlike the atomic APIs of earlier tool-use work (Schick et al., 2023; Patil et al., 2023; Qin et al., 2023), these packages contain their own workflow, resources, and executable operations, and agentic systems collect them in repositories and libraries to retrieve and reuse across tasks (Wang et al., 2023; Xu & Yan, 2026; Liang et al., 2026; Wang et al., 2026a; Lu et al., 2026b).

Before an agent starts on a task, it must decide which skills the task requires. Skill selection can be framed as a retrieval problem: matching a task instruction to the capabilities it requires. However, it resembles query–document retrieval with two differences: the candidates are themselves executable artifacts, and the queries are full task instructions with their own constraints. Neural retrieval work has shown that representation quality is central to the matching (Reimers & Gurevych, 2019; Karpukhin et al., 2020; Thakur et al., 2021). Tool- and skill-retrieval work further shows that capability matching depends on signals scattered across interface fields, instructional text, examples, implementation details, and structural dependencies, and that generic retrievers do not always transfer reliably to tool or skill selection (Yuan et al., 2024; Shi et al., 2025; Zheng et al., 2024; Lin et al., 2025; Zheng et al., 2026; Li et al., 2025). Recent systems organize skill collections themselves as graphs to scale selection and composition (Liu et al., 2026; Bai et al., 2026; Yao et al., 2026), and work on tool-document compression and enrichment argues that verbose tool descriptions often need to be reorganized for effective retrieval (Yuan et al., 2025; Lu et al., 2026a).

Once a skill is selected, what follows is how the agent turns it into action: interpreting instructions, following steps, using resources, and adapting to feedback. Existing work studies this process through trajectory-induced procedures (Wang et al., 2024), gated workflows (El Hattami et al., 2026), runtime route revision (Wang et al., 2026b), and transfer evaluation (Belikova et al., 2026). A second line asks how the skill is exposed to the agent at all (Ren et al., 2026), with presentation granularity tied to downstream success (Xu & Wu, 2026).

Across both phases, the skill itself is taken as given: stored, learned, routed, presented, or induced, while its internal form goes unexamined. Retrieval treats each skill's representation as already available, and execution treats its instructions as directly followable. The question left open is how an existing skill should be restructured into a machine-usable form that disentangles invocation interface, scene-level execution structure, and action/resource-use evidence, both to sharpen retrieval before a task and to steer execution during it.

2.2 Cognitive Linguistic Representations of Activities

Cognitive linguistic representation studies how activities can be organized beyond surface text. Within Schank and Abelson's line of classical work, Memory Organization Packets (MOPs) model recurring goal-oriented contexts (Schank, 1980); Script Theory (ST) represents stereotyped activities as ordered event sequences with roles and transitions (Schank & Abelson, 1977); Conceptual Dependency (CD) decomposes linguistic meaning into primitive action structures (Schank, 1972). Related frame-based theories make a similar commitment to structured context: Minsky's Frame Theory represents familiar situations through slots, defaults, and expectations (Minsky, 1975), and Fillmore's Frame Semantics treats meaning as grounded in scenes with participant roles (Fillmore, 1982).

While frame-based theories may support the broader premise that linguistic content can be organized around context, Schank and Abelson's work provides a closer analogy for SSL by distinguishing

goal-oriented contexts, ordered activities, and primitive operations, aligning closely with the three kinds of evidence SSL disentangles: invocation-level interfaces, scene-level execution structure, and atomic action/resource-use evidence. Appendix C summarizes CD, ST, and MOPs for readers unfamiliar with this literature.

3 The Scheduling-Structural-Logical Representation of Agent Skills

SSL represents a skill by disentangling three kinds of source-grounded evidence: when the skill should be invoked, how its work is organized, and what operations or resources it may involve. This decomposition is inspired by Schank and Abelson's cognitive linguistic representation theories (Appendix C): MOPs motivate goal- and context-level capability records (Schank, 1980); ST motivates ordered execution phases with conditions and transitions (Schank & Abelson, 1977); and CD motivates primitive action structures with roles, effects, and resource targets (Schank, 1972). These theories serve as design analogies for a systems schema whose purpose is to make skill artifacts easier to manage, inspect, validate, act on, and reuse.

3.1 Problem Formulation and Design Goals

Let d denote a skill artifact, such as a `SKILL.md` file. SSL answers three questions about a skill: when an agent should invoke it, how its work is organized, and which actions and resources it involves. SSL maps d into a typed representation:

$$G_d = (r_{\mathrm{sch}}, G_{\mathrm{str}}, G_{\mathrm{log}}, R_{\mathrm{cont}}, R_{\mathrm{entry}}), \tag{1}$$

where r_{sch} (the *Scheduling* layer) records when the skill should be used and what it requires at invocation; G_{str} (the *Structural* layer) arranges its work into ordered scenes with transitions; G_{log} (the *Logical* layer) lists the concrete actions and the tools or data each action touches; R_{cont} links scenes to the skill and steps to scenes; and R_{entry} marks where traversal begins.

Operationally, r_{sch} is realized as a single skill-level record; scene records instantiate G_{str} as a directed graph over execution phases; and logic-step records instantiate G_{log} as a directed graph over atomic actions within those scenes. The complete field-level realization is listed in Appendix A.

SSL follows three design goals: it is *compact*, preserving evidence needed for skill management and use while avoiding open-ended or subjective attributes; it is *typed*, using restricted vocabularies so normalized outputs remain comparable across skills; and it is *grounded*: fields strictly summarize evidence present in the source artifact, making no attempt to infer hidden behavior. The intended scope follows from this grounding: SSL applies to artifacts whose operational content is carried by natural-language instructions with embedded commands, including `SKILL.md`-style documents; it neither assumes nor invents runtime behavior, and artifacts consisting of undocumented executable code fall outside its scope. Appendix B provides a compact complete instance.

3.2 The Three-Layer SSL Representation

As illustrated in Figure 1, we detail the three-layer SSL representation as follows.

3.2.1 Scheduling Layer: Skill-Level Interface

The scheduling layer corresponds to r_{sch} in Eq. 1. Skill registries already attach a free-text description to each skill for human browsing; the scheduling layer instead organizes the invocation surface as a typed record. It exposes supported intents, input/output contracts, and coarse dependencies or control-flow properties as distinct fields, so a consumer can compare skills across a repository without unfolding their full scene or logic-step structure.

3.2.2 Structural Layer: Scene-Level Execution Phases

The structural layer corresponds to G_{str} in Eq. 1. Its nodes are scenes, and its edges represent phase-level transitions among those scenes. Motivated by script-like event structure (Schank & Abelson, 1977), which we summarize in Appendix C, this layer groups low-level operations into

coherent stages, such as preparation, acquisition, reasoning, action and verification, making the skill's phase organization visible before inspecting individual logic steps.

3.2.3 Logical Layer: Atomic Actions and Resource Evidence

The logical layer corresponds to $G_{\log}$ in Eq. 1. Its nodes are logic steps, and its edges represent micro-level transitions among source-grounded atomic actions. Each atomic action selects an `act_type` from the closed primitive inventory as described in Appendix A, and records arguments, effects, and resource boundaries as typed evidence. Because atomicity is a property of the representation irrespective of runtime details, a logic step is simply the smallest operational unit the source artifact supports without inventing missing implementation details.

3.3 The SSL Normalization Pipeline

SSL is instantiated with an LLM-based normalizer that converts the full source document of a skill artifact into the SSL representation. Operating strictly as a semantic extractor, the normalizer avoids open-ended summarization: its prompt specifies the schema, allowed vocabularies, and grounding policy, and every populated field must be supported by the source artifact. As summarized in Appendix D, the pipeline extracts the skill-level record, decomposes the document into scenes, and expands each scene into source-grounded logic steps. Before a record reaches any downstream experiment, it must pass a hard validator, which checks parseability, required fields, identifier uniqueness, allowed enum values, containment links, entry pointers, and transition targets. Outputs that fail parsing or validation enter a repair loop: the validator's error list is appended to the prompt and the model revises its own output.

4 Evaluation

We evaluate SSL in two aspects. The first is Skill Routing: before the agent works on a task, the system must select the skill or skills the task requires from a registry. Routing compares short candidate lists derived from the SSL view against those derived from the raw view under an identical reranker, so any difference comes from which candidates enter the short list. The second is Skill Execution: after selection, the agent performs the task with the workflows of the selected skills at hand, and the question is how the skill should be presented for the agent to act on it. Execution delivers the same passages from the task's skill files with or without SSL's relation labels, so any difference comes from the annotations on those passages.

4.1 Evaluation of Skill Routing

4.1.1 Benchmark and Protocol

We use the SkillRouter benchmark (Zheng et al., 2026), which provides 75 scored tasks with 200 gold skill labels in total. The hard tier contains 79,141 candidate skills and the easy tier 78,361. We select it because it evaluates Skill Routing over skill artifacts at registry scale, which is exactly the setting our representation targets.

Routing follows a two-stage pipeline, since a cross-encoder cannot score every candidate at this registry scale. In the first recall stage, a dense retriever (SkillRouter-Embedding-0.6B) embeds every candidate under two representations: the raw view (name, description, and full source text; 1,941 tokens per candidate on average) and the complete SSL view (serialized SSL fields; 1,566 tokens on average), and produces a short list of w candidates by cosine similarity. In the second stage, a cross-encoder reranker (SkillRouter-Reranker-0.6B) re-scores each short list.

The study is inference-only, and the two conditions share encoders, queries, reranker, and window, so the only difference is which view the retriever embeds. The short lists then differ only in composition, while the reranker scores the same raw text in both. Feeding SSL text to the reranker would form a different pipeline and is left open. Per-query costs are matched, and the compact view only shrinks the one-time index. We report nDCG@10, Recall@20, and Hit@1 at short-list windows $w \in \{20, 50, 100\}$. The SSL records for the candidate pool are generated from source documents alone by a single-pass normalizer (DeepSeek-V4.1-Flash; DeepSeek-AI et al., 2026).

		nDCG@10			Recall@20			Hit@1		
Tier	**Window**	Raw	SSL	Δ	Raw	SSL	Δ	Raw	SSL	Δ
Hard	20	0.4064	**0.4332**	+2.69	0.6032	**0.6515**	+4.8	0.5867	**0.6267**	+4.0
Hard	50	0.4228	**0.4360**	+1.32	0.6433	**0.6677**	+2.4	0.6000	**0.6400**	+4.0
Hard	100	0.4422	**0.4424**	+0.02	**0.7088**	0.6806	−2.8	0.6267	**0.6400**	+1.3
Easy	20	0.4257	**0.4551**	+2.95	0.6355	**0.6771**	+4.2	0.6133	**0.6667**	+5.3
Easy	50	0.4424	**0.4532**	+1.08	0.6610	**0.6877**	+2.7	0.6133	**0.6533**	+4.0
Easy	100	**0.4624**	0.4584	−0.40	**0.7115**	0.6906	−2.1	**0.6533**	**0.6533**	0.0

Table 1: Skill Routing on the SkillRouter benchmark. Window is the first-stage short-list size. nDCG@10 is the primary metric; Recall@20 is the fraction of gold skills in the top 20 of the short list (invariant to reranking), and Hit@1 is the rate at which the reranker's top pick is a gold skill. Δ is SSL minus Raw in points, computed from unrounded means. All values are means over the 75 scored tasks. Both conditions score identical candidate text, so differences reflect short-list composition. Bold marks the better value; ties are bolded in both.

View	**Tokens/cand.**	**nDCG@10**
`raw`	1,941	0.4034
`raw_head2048`	1,316	0.4087
`raw_match_ssl`	1,179	0.4105
`ssl`	1,566	**0.4287**

Table 2: Length controls at window 20 on the hard tier. `raw_match_ssl` truncates each candidate to the SSL view's own length, so it is never longer than `ssl`.

4.1.2 RESULTS

Table 1 reports the paired comparison. At window 20, replacing short lists derived from the raw view with short lists derived from the SSL view improves nDCG@10 by $+2.69$ pp on the hard tier and $+2.95$ pp on the easy tier, which is 6.6% and 6.9% relative to the raw baseline. The advantage concentrates at small windows and disappears by window 100. The set-level metrics agree: at window 20 the SSL short lists contain the gold skill more often (Recall@20 $+4.8$ pp hard, $+4.2$ pp easy), and the reranker's top pick is the gold skill more often (Hit@1 $+4.0$ pp hard, $+5.3$ pp easy). By window 100 the set-level advantage inverts on the hard tier, matching the disappearance of the ranked gain. This mirrors the first-stage measurements: SSL short lists are more precise at small candidate budgets, whereas at large budgets the raw ordering catches up; Appendix E reports the full window sweep.

Two registered control experiments test two rival explanations: the SSL view is shorter than the raw view, so the window-20 gain might be a length artifact, and if any restructuring helped equally, the three-layer design itself would not matter.

First, the gain is not plain compression. Because the SSL view is shorter than the raw view on average, we re-ran the window-20 comparison against length-matched raw controls, truncated per candidate to the SSL view's token budget. Table 2 reports the outcome: the SSL view stays ahead of the tightest control by $+1.82$ pp on the hard tier and $+2.85$ pp on the easy tier, and the matched control is the shorter view.

Second, the gain is mostly localized in the scheduling layer. We rank all 79,141 candidates with each ablation view and record how far down its ranking each task's gold skills fall (mean rank percentile, lower is better). Table 3 lists the key contrasts: the card beats a plain-prose condensation of the same document written at its own token budget (`summ_long`), and appending the graph layer to the card pushes the gold skills down, and the scheduling card stays $11.3\times$ shorter than the raw text it replaces. Full tables and serialization examples of the three views are reported in Appendix E.

The two stages split the work: reranking the top-20 short list lifts the SSL ordering by $+5.91$ pp and the raw ordering by $+2.10$ pp, and without the reranker the SSL ordering sits slightly below raw. The $+2.69$ pp in Table 1 is realized between the reranked conditions. Appendix I follows one task through both views at window 20.

View	Δ vs `card`	p
`summ_long`	$+0.62$	0.003
`sched_rel`	$+0.42$	0.0002
`sched_facts`	$+0.40$	0.0001
`raw`	$+0.24$	0.146

Table 3: Gold-rank percentile of each view minus the scheduling card's (`card`), over all 79,141 hard-tier candidates. Lower percentile is better, so a positive value means the card ranks the gold skills higher than that view. `summ_long` is prose written from the same document by an LLM, rendered at the card's token budget; `sched_rel` and `sched_facts` append the scene–logic relations and the per-action facts to the card. Constructions are listed with the other views in Appendix E.

	DeepSeek			GLM		
Condition	**Reward**	**Full**	**Calls**	**Reward**	**Full**	**Calls**
`raw`	0.7316	72.4%	27.6	0.5737	56.6%	22.4
`card_flat`	0.7184	71.1%	26.2	0.5342	52.6%	**19.3**
`card_ssl`	**0.7815**	**76.3%**	**25.6**	**0.6526**	**64.5%**	20.8
`card_ssl_rand`	0.7127	69.7%	29.4	0.5600	53.9%	21.5
`card_ssl_shuffle`	0.6912	68.4%	28.6	0.5958	56.6%	20.7

Table 4: SSL-guided Skill Execution on SkillsBench. Reward is the task's verified score, partial credit included; Full is the fraction of runs with full credit; Calls is the mean number of model calls per run. Bold marks the best condition in the primary run.

4.2 Evaluation of Skill Execution

4.2.1 Benchmark and Protocol

For Skill Execution, we experiment on SkillsBench (Li et al., 2026) to test how the SSL representation, delivered at execution time, affects success. We select it because its tasks pair each instruction with the skill documents the solution depends on and score outcomes with executable tests, so how the skill is delivered can be varied without changing the task. It is also among the most widely used benchmarks for skill use. How to deliver skills is a design question: the skills' own files remain in context, so the agent can already read them, and what the files do not expose is which parts matter for the task at hand and how they bind together. We therefore deliver the SSL structure as source-grounded prompt: the harness stages an action card into the agent's initial prompt that copies selected passages from the task's skill files, and annotates each passage with its SSL relation, in which each annotation is a logic step binding the action to its inputs, instrument, and expected result.

Five conditions test this design: `raw` only receives the skill; `card_flat` receives the selected passages word for word from the task's skill files, each with its file location, listed in the order they appear in the files; `card_ssl` receives exactly the same passages and locations, plus one annotation from SSL per passage, stating what the action takes as input, what it applies, and what result to check; `card_ssl_rand` and `card_ssl_shuffle` probe what carries the card's effect, while the former delivers a card built for a different task, and the latter pairs the annotations with the wrong passages. Each condition runs on the 76 eligible tasks with two models (DeepSeek-V4.1-Flash; DeepSeek-AI et al., 2026 and GLM-5.3-Flash; GLM-5-Team et al., 2026). The SSL records behind the cards are produced from the source skill files alone by the same single-pass normalizer used in routing (DeepSeek-V4.1-Flash). Appendix G details the card construction.

4.2.2 Results

Table 4 reports mean verified reward for each condition, separately for each agent model. The SSL-annotated card improves verified success over `raw` by $+5.00$ pp on DeepSeek (0.732 to 0.782) and $+7.89$ pp on GLM (0.574 to 0.653), which is 6.8% and 13.8% relative to `raw`. The gain sits in the annotations: `card_flat`, which delivers the same passages without annotations, is statistically indistinguishable from `raw`, 1.32 pp below on DeepSeek (0.718 versus 0.732) and 3.95 pp below on GLM (0.534 versus 0.574), while annotating those same passages improves success by $+6.31$ pp

Condition	Card Chars	Model Calls	Skill Reads
raw	—	25.0	1.64
card_flat	1,302	22.8	2.12
card_ssl	1,361	23.2	2.09

Table 5: Cost and engagement per condition, pooled over both models. Card Chars is the length of the text the condition adds to the prompt, Model Calls is the mean number of model calls per run, and Skill Reads is the mean number of the agent's reads of the task's own skill files per run.

on DeepSeek (0.718 to 0.782) and $+11.84$ pp on GLM (0.534 to 0.653). The card adds no cost: it is prompt text, and the SSL-annotated card uses fewer model calls than raw on both models (25.6 against 27.6 on DeepSeek, 20.8 against 22.4 on GLM). Table 5 reports the cost side in full: the two card conditions differ in length by 4.5%, so the contrast between them is length-matched, and neither card raises the call count while both raise the agent's reads of the task's own skill files.

5 Discussion

5.1 How SSL Benefits Skill Routing

In the routing pipeline, the view decides which candidates reach the short list and the reranker decides their order, so the view can only help by changing the composition of that list. Table 1 shows where it helps: the gain appears at small windows and disappears by window 100, which makes SSL a representation for candidate generation at small budgets. That is the budget a deployed agent actually reads, since it consumes a handful of candidate skills per request.

The two controls rule out the alternatives: the length-matched comparison excludes plain compression, and the remaining gain localizes in the scheduling fields. That localization is exactly what the consumer needs: selecting the right skill for a task is a decision about what each skill is for and what it requires at invocation, and the scheduling layer is the typed record of it. The deployment lesson is to project the representation per consumer: routing consumes the scheduling card alone, at the full view's quality and a fraction of its size.

5.2 How SSL Benefits Skill Execution

The scene–logic relations and per-action facts are meant for the agent at execution time. The execution experiment asks whether they help. On SkillsBench, the annotated card improves verified success over raw delivery on both agent models, while the same passages without annotations are statistically indistinguishable from raw delivery: the gain is carried by the annotations, which hold the logical layer's content in SSL. Each passage states what the action takes as input, what instrument it applies, and what result to verify. The card quotes source spans, and the annotated conditions read the skill's own source files more often than raw (Appendix G), so the card points the agent at the source while making the action structure explicit. Appendix F reports the two earlier delivery tries, SSL text in the prompt and a post-turn review message, and neither produced a reliable benefit.

What the result establishes is a consumer and a time: at execution, success changes with the form in which the skill reaches the agent, and the gain sits in the logical layer's annotations. Appendix I follows one task's three cards against its recorded runs.

5.3 How SSL Serves Skill-Centered Agentic Systems

Both experiments place SSL beside the source document, and neither uses it as a replacement for the source. Routing's deciding stage reads each candidate's full raw text, and execution's card enters the prompt beside the passages it quotes: in both cases the typed surface selects and organizes, while the source document remains the text the reranker scores and the files the agent reads.

Scaled from our two experiments, the pattern suggests a design choice: a skill keeps a typed record of its facts beside its source document. The reason is that a skill has many consumers: registries, routers, policy checkers, and reviewers each need the same facts, and today each re-derives them from the

same `SKILL.md` file, so a record stored once beside the file serves all of them. The routing results add an economic incentive: at registry scale (79,141 candidates in the hard tier alone), the scheduling card shrinks the embedding index by $11.3\times$ relative to raw text while improving small-window precision. The same record can also support skill construction and maintenance by making expected inputs, phase boundaries, dependencies, and resource effects explicit.

6 Conclusion

We introduced SSL, what is to our knowledge the first structured representation for agent skill artifacts that disentangles skill-level scheduling signals, scene-level execution structure, and logic-level action and resource evidence. We instantiated it with an LLM-based normalizer and evaluated it along two aspects: Skill Routing before a task, and Skill Execution during it. In Skill Routing, indexing candidate skills by their SSL views improves nDCG@10 over the raw view by $+2.69$ pp on the hard tier and $+2.95$ pp on the easy tier (6.6% and 6.9% relative). In Skill Execution, an action card that annotates passages of the task's skill files with SSL relations improves verified task success over raw execution by $+5.00$ pp on DeepSeek-V4.1-Flash and $+7.89$ pp on GLM-5.3-Flash (6.8% and 13.8% relative) at no additional model calls. Together, the two aspects show that an explicit, source-grounded SSL representation helps both when structure selects what the agent sees and when it governs how the agent acts.

Future work runs in three directions: For the existing SSL representation, the structural layer still awaits the consumers it was written for, such as runtimes that deliver scenes as a task progresses and monitors that check a run against its phase gates; For a better representation, runtime traces and post-execution feedback can enrich normalization, so that a record keeps pace with the skill it describes through authoring, updates, and reuse; For agentic systems, integration can reach beyond the selection and delivery stages we tested, covering planning, prompt construction, and verification, with every stage reading the same facts. We position SSL as a practical step toward more inspectable, reusable, and operationally actionable skill representations, and as a building block for future skill registries and agent runtimes.

7 Limitations

The study has several limitations at the current stage:

- **Static extraction**: SSL is extracted from static artifacts and cannot determine dynamic behavior by itself, such as downloaded payloads, constructed commands, or conditional resource access. Controllers staged from SSL workflows inherit this boundary;
- **Normalizer fidelity**: SSL extraction depends on LLM-based normalization, which may omit relevant facts, over-regularize the source, or map ambiguous behavior into coarse enums for underspecified or obfuscated skills;
- **Single benchmark family**: Routing is evaluated on a single benchmark (SkillRouter) with one reranker and two embedding encoders, and execution on a single benchmark (SkillsBench). Neither result establishes transfer to other task families, skill formats, or runtimes. For routing, we do not test SSL-structured text as the reranker's input: the reranker reads raw text under its own input contract, and the counterfactual pipeline with a structure-aware scorer is left open;
- **Window-dependent advantage**: The routing advantage concentrates at small windows and disappears by window 100, with the set-level comparison inverting on the hard tier. Claims should be scoped to candidate-generation budgets of this size;
- **Structural layer consumers**: The routing benchmark reads the scheduling layer and the execution card carries the logical layer, so the scene graph of phases and transitions is written for consumers this paper does not evaluate: orchestrators that sequence skills, runtimes that deliver scenes, and phase-gate monitors.

AI USE STATEMENT

Generative AI appears in this work in two distinct roles. As systems under study, large language models drive both evaluation axes: the SSL normalizer (DeepSeek-V4.1-Flash) converts skill artifacts into structured records, and the routing and execution experiments are run with LLM agents (Sections 4.1 and 4.2; Appendices D, E, and G). As paper assistance, we used generative AI tools to aid and polish writing, to draft sections of the paper, for retrieval and discovery of related work, and for research ideation and execution, including developing and running analysis scripts over the registered experiment artifacts; details are described in the paper. We have not used generative AI tools for authoring tasks beyond these uses. We have reviewed all AI-assisted work: every reported number was read directly from the registered experiment records, and every AI-assisted passage was checked and revised by the authors. The models did not replace author judgment in the interpretation of results or the final claims. The authors take full responsibility for the final content, claims, and artifacts of this work, including text and code produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The normalization protocol, benchmark construction, evaluation settings, and bootstrap procedure are described in the main text and Appendix H. The released artifacts are the SSL schema, the normalizer prompt, and the minimal supporting code for record validation, view serialization, and action-card construction; normalized records and per-run artifacts are not part of the release, and the evaluation builds on the public SkillRouter Benchmark and SkillsBench.

REFERENCES

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. URL https://arxiv.org/abs/2606.03056.

Julia Belikova, Rauf Parchiev, Evgeny Egorov, et al. Managing procedural memory in LLM agents: Control, adaptation, and evaluation. *arXiv preprint arXiv:2606.23127*, 2026. URL https://arxiv.org/abs/2606.23127.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5): 34–43, 2001. URL https://www.scientificamerican.com/article/the-semantic-web/.

DeepSeek-AI, Anyi Xu, B. Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, et al. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026. doi: 10.48550/arXiv.2609.19969. URL https://arxiv.org/abs/2609.19969.

Amine El Hattami, Nicolas Chapados, and Christopher Pal. SKILL.nb: Selective formalization and gated execution for durable agent workflows. *arXiv preprint arXiv:2606.08049*, 2026. URL https://arxiv.org/abs/2606.08049.

Charles J. Fillmore. Frame semantics. In *Linguistics in the Morning Calm*, pp. 111–137. Hanshin Publishing Company, Seoul, 1982.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, et al. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. doi: 10.48550/arXiv.2602.15763. URL https://arxiv.org/abs/2602.15763.

César González-Mora, Cristina Barros, Irene Garrigós, Jose Zubcoff, Elena Lloret, and Jose-Norberto Mazón. Improving open data web API documentation through interactivity and natural language generation. *Computer Standards & Interfaces*, 83:103657, 2023. doi: 10.1016/j.csi.2022.103657. URL https://doi.org/10.1016/j.csi.2022.103657.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Koren Lazar, Matan Vetzler, Kiran Kate, Jason Tsay, David Boaz, Himanshu Gupta, Avraham Shinnar, Rohith D. Vallam, David Amid, Esther Goldbraich, Guy Uziel, Jim Laredo, and Ateret Anaby Tavor. Generating OpenAPI specifications from online API documentation with large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 237–253, 2025. doi: 10.18653/v1/2025.acl-industry.18. URL https://aclanthology.org/2025.acl-industry.18/.

Fangzhou Li, Pagkratios Tagkopoulos, and Ilias Tagkopoulos. SkillFlow: Scalable and efficient agent skill retrieval system. *arXiv preprint arXiv:2504.06188*, 2025. URL https://arxiv.org/abs/2504.06188.

Xiangyi Li, Yimin Liu, Wenbo Chen, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026. URL https://arxiv.org/abs/2602.12670.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. SkillNet: Create, evaluate, and connect AI skills. *arXiv preprint arXiv:2603.04448*, 2026. doi: 10.48550/arXiv.2603.04448. URL https://arxiv.org/abs/2603.04448.

Jianghao Lin, Xinyuan Wang, Xinyi Dai, Menghui Zhu, Bo Chen, Ruiming Tang, Yong Yu, and Weinan Zhang. MassTool: A multi-task search-based tool retrieval framework for large language models. *arXiv preprint arXiv:2507.00487*, 2025. doi: 10.48550/arXiv.2507.00487. URL https://arxiv.org/abs/2507.00487.

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

Xuan Lu, Haohang Huang, Rui Meng, Yaohui Jin, Wenjun Zeng, and Xiaoyu Shen. Tools are under-documented: Simple document expansion boosts tool retrieval. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=g9D9MgG7iW.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026b. doi: 10.48550/arXiv.2604.02268. URL https://arxiv.org/abs/2604.02268.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, Yiqiao Jin, Fan Zhang, Xian Wu, Hanqing Zhao, Dacheng Tao, Philip S. Yu, and Ming Zhang. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025. doi: 10.48550/arXiv.2503.21460. URL https://arxiv.org/abs/2503.21460.

Marvin Minsky. A framework for representing knowledge. In Patrick H. Winston (ed.), *The Psychology of Computer Vision*, pp. 211–277. McGraw-Hill, New York, 1975.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023. doi: 10.48550/arXiv.2305.15334. URL https://arxiv.org/abs/2305.15334.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023. doi: 10.48550/arXiv.2307.16789. URL https://arxiv.org/abs/2307.16789.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Shuo Ren, Xiaomian Kang, and Jiajun Zhang. SkillAlign: Aligning skill interfaces for LLM-based agents. *arXiv preprint arXiv:2609.07255*, 2026. URL https://arxiv.org/abs/2609.07255.

Roger C. Schank. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, 3(4):552–631, 1972. doi: 10.1016/0010-0285(72)90022-9.

Roger C. Schank. Language and memory. *Cognitive Science*, 4(3):243–284, 1980. doi: 10.1207/s15516709cog0403_2.

Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, Cambridge, 1982.

Roger C. Schank and Robert P. Abelson. Scripts, plans, and knowledge. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pp. 151–157, Tbilisi, USSR, 1975.

Roger C Schank and Robert P Abelson. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. L. Erlbaum, 1977.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24497–24524, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1258. URL https://aclanthology.org/2025.findings-acl.1258/.

Nandan Thakur, Nils Reimers, Andreas Rucklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL https://openreview.net/forum?id=wCu6T5xFjeJ.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026a. doi: 10.48550/arXiv.2604.04804. URL https://arxiv.org/abs/2604.04804.

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. doi: 10.48550/arXiv.2305.16291. URL https://arxiv.org/abs/2305.16291.

Tianxing Wang, Mingming Zhao, Shuai Huang, Huiyang Xu, Chaoyue Niu, Shengzhong Liu, and Fan Wu. TROVE: Adaptive agent skill orchestration via trace-grounded route validation and editing. *arXiv preprint arXiv:2609.05019*, 2026b. URL https://arxiv.org/abs/2609.05019.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. URL https://arxiv.org/abs/2409.07429.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023. doi: 10.48550/arXiv.2309.07864. URL https://arxiv.org/abs/2309.07864.

Haoyuan Xu, Chang Li, Xinyan Ma, Xianhao Ou, Zihan Zhang, Tao He, Xiangyu Liu, Zixiang Wang, Jiafeng Liang, Zheng Chu, Runxuan Liu, Rongchuan Mu, Dandan Tu, Ming Liu, and Bing Qin. The evolution of tool use in llm agents: From single-tool call to multi-tool orchestration. *arXiv preprint arXiv:2603.22862*, 2026. doi: 10.48550/arXiv.2603.22862. URL https://arxiv.org/abs/2603.22862.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026. doi: 10.48550/arXiv.2602.12430. URL https://arxiv.org/abs/2602.12430.

Xiaonan Xu and Wenjing Wu. Skill availability and presentation granularity in large-language-model agents: A controlled SkillsBench study. *arXiv preprint arXiv:2605.31408*, 2026. URL https://arxiv.org/abs/2605.31408.

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. URL https://arxiv.org/abs/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*, 2024. doi: 10.48550/arXiv.2309.17428. URL https://arxiv.org/abs/2309.17428.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 951–972, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.44. URL https://aclanthology.org/2025.naacl-long.44/.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. doi: 10.48550/arXiv.2506.05176. URL https://arxiv.org/abs/2506.05176.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. SkillRouter: Skill routing for LLM agents at scale. *arXiv preprint arXiv:2603.22455*, 2026. doi: 10.48550/arXiv.2603.22455. URL https://arxiv.org/abs/2603.22455.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. ToolRerank: Adaptive and hierarchy-aware reranking for tool retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 16263–16273, Torino, Italia, 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.1413/.

A Definition of SSL Schema

This appendix gives the field-level realization of the SSL representation defined in Eq. 1. We first state the graph-design principles behind the schema, then describe the scheduling, structural, and logical layers and their restricted vocabularies.

A.1 Graph-Design Principles

The SSL schema realizes Eq. 1 as a typed JSON graph with three linked representational levels. The Scheduling Layer is a single skill-level record that stores the interface information needed for selecting and invoking the skill. The Structural Layer is a directed scene graph whose nodes are execution phases and whose edges encode phase-level transitions. The Logical Layer is a directed logic-step graph whose nodes are source-grounded atomic actions and whose edges encode action-level control flow within a scene. Cross-level connectivity is deliberately restricted to containment

Layer	Principal Fields	Purpose

Scheduling	`skill_id`
	`skill_name`
	`skill_goal`
	`intent_signature`
	`tags`
	`top_pattern`
	`expected_inputs`
	`expected_outputs`
	`dependencies`
	`control_flow_features`
	`entry_scene_id`
	`subscenes`
		Exposes the skill's routing and invocation interface without requiring downstream systems to embed or parse the full source document.

Structural	`scene_id`
	`scene_name`
	`scene_type`
	`scene_goal`
	`input`
	`output`
	`entry_conditions`
	`exit_conditions`
	`next_scene_rules`
	`entry_logic_step_id`
	`contained_logic_steps`
		Represents the skill as a sequence or graph of typed execution phases, making multistep workflows inspectable.

Logical	`logic_step_id`
	`act_type`
	`actor`
	`object`
	`instrument`
	`input_args`
	`output_binding`
	`preconditions`
	`effects`
	`resource_scope`
	`resource_target`
	`next_step_rules`
		Records bounded action and resource-use facts that are useful for data-flow and resource inspection.

Table 6: Core fields of the SSL schema.

relations and entry pointers: scenes belong to a skill, logic steps belong to scenes, and traversal begins from designated entry records. This restriction preserves clear boundaries among interface evidence, phase structure, and action/resource-use evidence while keeping cross-level relations explicit and limited.

The schema is designed to be *compact*, *typed*, and *grounded*. *Compactness* bounds what a record can hold: skill-level interface contracts and dependencies, scene-level phase structure, and logic-level data-flow cues, resource boundaries, and operational effects, while excluding open-ended attributes such as subjective skill quality, user persona, inferred developer intent, or speculative hidden behavior. *Typing* makes records comparable across skills: every value comes from the restricted vocabularies in Table 7. *Grounding* keeps records checkable against the file: each logic step carries the source spans it summarizes. Table 6 summarizes the principal fields for each layer, and the following subsections explain how these fields and vocabularies are used.

A.2 Scheduling Layer

The scheduling layer stores the skill-level interface. Its role is to expose what a skill is for and how it can be invoked without requiring downstream systems to re-embed or re-parse the entire source document. Fields such as `skill_goal`, `intent_signature`, `tags`, `top_pattern`, `expected_inputs`, `expected_outputs`, and `dependencies` capture the stable capability-facing surface of the skill,

Field or Constraint	Allowed Values
scene_type	PREPARE, ACQUIRE, REASON, ACT, VERIFY, RECOVER, FINALIZE
act_type	READ, SELECT, COMPARE, VALIDATE, INFER, WRITE, UPDATE_STATE, CALL_TOOL, REQUEST, TRANSFER, NOTIFY, TERMINATE
resource_scope	MEMORY, LOCAL_FS, CODEBASE, PROCESS, USER_DATA, CREDENTIALS, NETWORK, OTHER
target terminal values	END_SUCCESS, END_FAIL, YIELD_SUCCESS, YIELD_FAIL

Table 7: Restricted vocabularies used by the SSL normalizer.

while entry_scene_id and subscenes connect that surface to the execution graphs below. To avoid duplicating the full execution graph at the scheduling level, the top-level control_flow_features field remains intentionally coarse, storing summary signals such as whether the skill contains branching, loops, tool calls, or sensitive-resource access.

A.3 Structural Layer

The structural layer represents a skill as a scene-level execution graph. Instead of an arbitrary text span or sentence block, a scene should denote a coherent execution phase with its own goal, data contract, and exit conditions. Accordingly, fields such as scene_type, scene_goal, input, output, entry_conditions, exit_conditions, and next_scene_rules describe how the skill unfolds at the phase level. Control flow is represented here through next_scene_rules, whose nested target field must either name another scene in the same graph or use a reserved terminal value. END_SUCCESS and END_FAIL therefore act as closed control-flow outcomes for the scene graph, not free-form labels; together with the logic-level YIELD_SUCCESS and YIELD_FAIL values, they form the terminal-target vocabulary listed in Table 7. The scene_type field is likewise drawn from a closed inventory so that phase categories remain comparable across skills; as listed in Table 7, these types are PREPARE, ACQUIRE, REASON, ACT, VERIFY, RECOVER, and FINALIZE.

A.4 Logical Layer

The logical layer represents the operations that implement each scene. It is defined over source-grounded atomic actions, where "atomic" means the smallest operational unit that the source artifact supports without inventing missing implementation detail. A logic step should usually be split when the source supports a change in action type, resource boundary, effect, or control-flow outcome; for example, it is preferable to represent reading a local file, calling an external tool, and writing back to the codebase as distinct steps when those operations are separately recoverable from the artifact. Fields such as act_type, actor, object, instrument, input_args, output_binding, preconditions, effects, resource_scope, resource_target, and next_step_rules describe this action/resource-use evidence. Data-flow variables start with a prefix $, while unprefixed strings denote literal labels, resources, or natural-language conditions. Within a scene, next_step_rules route among logic steps; their targets must either name another in-scope step or use YIELD_SUCCESS or YIELD_FAIL to return control to the enclosing scene.

As listed in Table 7, the logical layer uses two closed vocabularies. The act_type field records the action primitive: READ for information access; SELECT, COMPARE, VALIDATE, and INFER for decision and transformation steps; WRITE and UPDATE_STATE for mutation; CALL_TOOL, REQUEST, TRANSFER, and NOTIFY for external interaction or data movement; and TERMINATE for explicit control termination. The resource_scope field separately records the operational boundary touched by the action. Both vocabularies are intentionally coarse, preventing idiosyncratic labels while preserving distinctions useful for comparing execution behavior, resource contact, and risk-relevant operations.

B A Complete Example of SSL

The following compact instance illustrates what a normalized SSL object looks like after annotation.

```
{
  "skill": {
    "skill_id": "SKILL_WRITING_REFINER",
    "skill_name": "Writing Refiner",
    "skill_goal": "Revise user-provided text for clarity and concision.",
    "top_pattern": "GUIDE_AND_APPLY",
    "expected_inputs": [
      {"name": "draft_text", "type": "str"},
      {"name": "writing_context", "type": "str"}
    ],
    "expected_outputs": [
      {"name": "revised_text", "type": "str"},
      {"name": "applied_principles", "type": "list"}
    ],
    "dependencies": [
      {"type": "permission", "value": "filesystem.read"},
      {"type": "capability", "value": "text_processing"}
    ],
    "tags": ["writing", "editing", "documentation"],
    "intent_signature": ["improve this text", "edit for concision"],
    "control_flow_features": {
      "has_branch": true,
      "has_loop": false,
      "has_tool_call": true,
      "touches_sensitive_resources": false
    },
    "entry_scene_id": "S_PREPARE",
    "subscenes": ["S_PREPARE", "S_ACQUIRE", "S_REVISE"]
  },
  "scenes": [
    {
      "scene_id": "S_PREPARE",
      "scene_name": "Prepare request",
      "scene_type": "PREPARE",
      "scene_goal": "Validate the request and infer the editing intent.",
      "input": ["$draft_text", "$writing_context"],
      "output": ["$parsed_intent", "$target_principles"],
      "entry_conditions": ["skill_dispatched"],
      "exit_conditions": ["writing_task_clarified"],
      "entry_logic_step_id": "L_VALIDATE_INPUT",
      "contained_logic_steps": ["L_VALIDATE_INPUT", "L_PARSE_CONTEXT"],
      "next_scene_rules": [
        {"condition": "success", "target": "S_ACQUIRE"},
        {"condition": "default", "target": "END_FAIL"}
      ]
    },
    {
      "scene_id": "S_ACQUIRE",
      "scene_name": "Acquire style guidance",
      "scene_type": "ACQUIRE",
      "scene_goal": "Load the style guidance needed for the task.",
      "input": ["$writing_context", "$target_principles"],
      "output": ["$loaded_guidelines"],
      "entry_conditions": ["writing_task_clarified"],
      "exit_conditions": ["guidelines_loaded"],
      "entry_logic_step_id": "L_SELECT_GUIDE",
      "contained_logic_steps": ["L_SELECT_GUIDE", "L_READ_GUIDE"],
      "next_scene_rules": [
        {"condition": "success", "target": "S_REVISE"},
        {"condition": "default", "target": "END_FAIL"}
      ]
    },
    {
      "scene_id": "S_REVISE",
      "scene_name": "Revise draft",
      "scene_type": "REASON",
      "scene_goal": "Apply the selected rules and return the revision.",
      "input": ["$draft_text", "$loaded_guidelines", "$parsed_intent"],
      "output": ["$revised_text", "$applied_principles"],
      "entry_conditions": ["guidelines_loaded"],
      "exit_conditions": ["text_revised", "summary_generated"],
      "entry_logic_step_id": "L_PARSE_GUIDE",
      "contained_logic_steps": [
        "L_PARSE_GUIDE", "L_SELECT_RULES", "L_APPLY_EDITING"
      ],
      "next_scene_rules": [
        {"condition": "success", "target": "END_SUCCESS"},
        {"condition": "default", "target": "END_FAIL"}
      ]
    }
  ],
```

```
"logic_steps": [
  {
    "logic_step_id": "L_VALIDATE_INPUT",
    "act_type": "VALIDATE",
    "input_args": ["$draft_text", "$writing_context"],
    "output_binding": "$input_valid",
    "actor": "skill",
    "object": "user_input",
    "instrument": null,
    "preconditions": ["skill_dispatched"],
    "effects": ["$input_valid == true"],
    "resource_scope": "MEMORY",
    "resource_target": "working_memory.user_request",
    "next_step_rules": [
      {"condition": "$input_valid == true", "target": "L_PARSE_CONTEXT"},
      {"condition": "default", "target": "YIELD_FAIL"}
    ]
  },
  {
    "logic_step_id": "L_PARSE_CONTEXT",
    "act_type": "INFER",
    "input_args": ["$writing_context"],
    "output_binding": "$target_principles",
    "actor": "skill",
    "object": "writing_context",
    "instrument": null,
    "preconditions": ["$input_valid == true"],
    "effects": ["writing_task_clarified"],
    "resource_scope": "MEMORY",
    "resource_target": "working_memory",
    "next_step_rules": [
      {"condition": "always", "target": "YIELD_SUCCESS"}
    ]
  },
  {
    "logic_step_id": "L_SELECT_GUIDE",
    "act_type": "SELECT",
    "input_args": ["$writing_context"],
    "output_binding": "$guide_file_path",
    "actor": "skill",
    "object": "guide_repository",
    "instrument": null,
    "preconditions": ["writing_task_clarified"],
    "effects": ["primary_guide_selected"],
    "resource_scope": "MEMORY",
    "resource_target": "guide_index",
    "next_step_rules": [
      {"condition": "always", "target": "L_READ_GUIDE"}
    ]
  },
  {
    "logic_step_id": "L_READ_GUIDE",
    "act_type": "READ",
    "input_args": ["$guide_file_path"],
    "output_binding": "$loaded_guidelines",
    "actor": "skill",
    "object": "style_guide_file",
    "instrument": "filesystem.read",
    "preconditions": ["primary_guide_selected"],
    "effects": ["guidelines_loaded"],
    "resource_scope": "LOCAL_FS",
    "resource_target": "$guide_file_path",
    "next_step_rules": [
      {"condition": "success", "target": "YIELD_SUCCESS"},
      {"condition": "default", "target": "YIELD_FAIL"}
    ]
  },
  {
    "logic_step_id": "L_PARSE_GUIDE",
    "act_type": "INFER",
    "input_args": ["$loaded_guidelines"],
    "output_binding": "$parsed_rules",
    "actor": "skill",
    "object": "guidelines",
    "instrument": null,
    "preconditions": ["guidelines_loaded"],
    "effects": ["rules_available"],
    "resource_scope": "MEMORY",
    "resource_target": "working_memory",
    "next_step_rules": [
      {"condition": "always", "target": "L_SELECT_RULES"}
```

```
      ]
    },
    {
      "logic_step_id": "L_SELECT_RULES",
      "act_type": "SELECT",
      "input_args": ["$parsed_rules", "$target_principles"],
      "output_binding": "$selected_rules",
      "actor": "skill",
      "object": "rule_set",
      "instrument": null,
      "preconditions": ["rules_available"],
      "effects": ["relevant_rules_selected"],
      "resource_scope": "MEMORY",
      "resource_target": "working_memory",
      "next_step_rules": [
        {"condition": "always", "target": "L_APPLY_EDITING"}
      ]
    },
    {
      "logic_step_id": "L_APPLY_EDITING",
      "act_type": "INFER",
      "input_args": ["$draft_text", "$selected_rules"],
      "output_binding": "$revised_text",
      "actor": "skill",
      "object": "draft_text",
      "instrument": "text_processing",
      "preconditions": ["relevant_rules_selected"],
      "effects": ["text_revised", "$applied_principles generated"],
      "resource_scope": "MEMORY",
      "resource_target": "working_memory.draft_text",
      "next_step_rules": [
        {"condition": "success", "target": "YIELD_SUCCESS"},
        {"condition": "default", "target": "YIELD_FAIL"}
      ]
    }
  ]
}
```


C A Primer on Schank's Cognitive Linguistic Representation Theories

The representation design of SSL draws on three classical theories from cognitive science and early artificial intelligence: Conceptual Dependency (CD), Script Theory (ST), and Memory Organization Packets (MOPs). This appendix summarizes each theory on its own terms.

C.1 Conceptual Dependency (CD)

Conceptual Dependency (CD) is a theory of natural language understanding introduced by Roger Schank (Schank, 1972). Its goal is to represent meaning independently of the particular words used, so that sentences identical in meaning share a single internal representation. CD decomposes meaning into a small inventory of primitive acts, together with real-world objects with attributes, actions, times, and locations. The primitive acts include ATRANS, the transfer of an abstract relationship, as in giving; PTRANS, the transfer of an object's physical location, as in going; MTRANS, the transfer of information between minds, as in telling; PROPEL, the application of physical force to an object; GRASP, the grasping of an object by an actor; MOVE, the movement of a body part by its owner; and INGEST and MBUILD, which appear in Schank's script analyses as taking in and mentally constructing. Under this scheme, a sentence such as "John gave a book to Mary" is represented as an ATRANS between John and Mary.

C.2 Script Theory (ST)

Script Theory (ST), developed by Schank and Abelson (Schank & Abelson, 1975; 1977), represents knowledge about familiar situations that recur. A script is a "predetermined, stereotyped sequence of actions" that defines a well-known situation. Formally, it is a structure of "slots and requirements": each slot states what may fill it, and the filler chosen for one slot narrows what can fill the ones that follow. Scripts resemble Minsky's frames, except that the values filling the slots must be ordered. The classic example is a restaurant script: the roles include customer, waitress, chef, and cashier; the

props include tables, menus, food, and money; and the event sequence is divided into scenes, the first of which, entering, contains acts such as PTRANS-ing oneself into the restaurant and ATTEND-ing to where the empty tables are. The acts within a scene are tied together by causal chaining: each action results in conditions that enable the next to occur, the previous acts must be completed satisfactorily before the next can proceed, and when they cannot be, the resulting hitches must be dealt with. Scripts also support expectation-driven understanding: a text only needs to invoke the script and mention a few distinctive events, since the script supplies the unstated events, and a typical story is a script with one or more interesting deviations. The SAM program demonstrated that short stories can be understood this way.

C.3 Memory Organization Packets (MOPs)

Memory Organization Packets (MOPs), introduced by Roger Schank (Schank, 1980; 1982), serve as both processors and organizers of information in memory: by categorizing experiences in episodic memory, they support better predictive understanding of new experiences. Where a script is a single fixed structure, a MOP organizes memory into smaller combinable modules that can be assembled as the situation requires. This line of work received little attention after the 1980s but became influential in later knowledge-representation research, most notably case-based reasoning.

D Prompting Protocol of the SSL Normalizer

This appendix documents the prompt contract shared by all instantiations and the generation configuration used for the skill-routing corpus.

D.1 Prompt Contract

Each request pairs a system message with a user message. The system message fixes the extraction contract:

> Convert one skill document into compact canonical SSL JSON. Use only the supplied candidate, never task queries, relevance labels, rankings, or outside knowledge. Output one JSON object only, with no markdown. Preserve the exact supplied `skill_id`.

The user message appends the schema specification, followed by the skill identifier, name, description, and the complete source body. Table 8 states the specification, and the grounding rule closes the contract:

> Do not invent tools, inputs, outputs, conditions, or workflows absent from the source. Use empty arrays for unsupported optional lists.

D.2 Validation and Repair Loop

Hard validation requires parseable JSON, all required fields, globally unique identifiers, valid enum values, valid containment links, valid entry pointers, and transition targets that name an in-scope node or a reserved terminal symbol. The retrieval projection used by the routing experiment additionally requires every `preconditions` element to be a string. Outputs that fail parsing or hard validation enter a repair loop: the previous JSON and the validator's error list are appended to the prompt, and the model revises its own output. Softer semantic checks, such as whether scene outputs are supported by logic-step bindings, guide repair and quality control; strict rejection remains the job of hard validation. If a field cannot be grounded after retries, it is left empty under a `null` convention recorded in the appendix contract.

Part	Specification
Record shape	exactly `{"skill": object, "scenes": array, "logic_steps": array}`, with 1–4 scenes and 1–6 logic steps in total
Skill	`skill_id` (preserved verbatim), `skill_name`, `skill_goal`, `intent_signature`, `expected_inputs`, `expected_outputs`, `dependencies`, `entry_scene_id`, and `subscenes`
Scene	`scene_id`, `scene_goal`, `contained_logic_steps`, `entry_logic_step_id`, `next_scene_rules`, and `scene_type` from the scene vocabulary; `entry_conditions` and `exit_conditions` are arrays of strings when present; each `next_scene_rules` item carries a condition and a target naming an emitted scene id, `END_SUCCESS`, or `END_FAIL`
Logic step	`logic_step_id`, `description`, `preconditions`, `effects`, `next_step_rules`, `act_type` from the action vocabulary, and `resource_scope` from the resource vocabulary or null; targets name an emitted step id, `YIELD_SUCCESS`, or `YIELD_FAIL`; every step is contained by exactly one scene

Table 8: Schema specification carried by the normalizer prompt. The closed vocabularies are those of Table 7.

E Skill Routing: Additional Controls and Analyses

This section reports in full the two routing controls that Section 4.1 summarizes, together with the serialization examples they use.

E.1 Budget-Matched Controls

The raw view is longer than the SSL view on average, so the window-20 advantage could in principle be a length effect. We test that explanation here: we materialize six candidate views with the same tokenizer and encoder and re-score all of them in a single cross-encoder run. The six views are built as follows. `raw` is the name, description, and full source body, capped at the encoder's 8,192-token limit (1.7% of candidates truncate). `raw_head512` and `raw_head2048` truncate the raw document from the head to 512 and 2,048 tokens. They are deliberately weak length controls, since document heads tend to carry titles and usage notes. `raw_match_ssl` truncates each raw document to that candidate's SSL length, which is the tightest control, always no longer than the SSL view. `ssl` and `schedule` are the standard views defined by the normalizer contract (Appendix D). Truncation is verified by re-encoding, since tokenization is sensitive to the surrounding text: a document prefix tokenized on its own can take more tokens than it started with. Each slice is shrunk until the re-encoded length fits its budget. Table 9 reports each view's budget, its average token count, and its window-20 nDCG@10 on the hard tier.

View	Budget	Tokens/cand.	nDCG@10 (w=20, hard)
`raw`	8,192 encoder cap	1,941.4	0.4034
`raw_head512`	512 tokens	471.5	0.3935
`raw_head2048`	2,048 tokens	1,315.8	0.4087
`raw_match_ssl`	per-candidate $\leq$ SSL	1,178.7	0.4105
`ssl`	2,048 tokens	1,566.3	**0.4287**
`schedule`	512 tokens	172.4	**0.4305**

Table 9: Budget-matched views on the hard tier; bold marks the two SSL views. Values come from a separate control run; its Raw value differs slightly from the main run because the cross-encoder is re-run here, and all comparisons are within this run.

Table 10 reports the paired differences against the SSL arm, computed on the hard tier at windows 20, 50, and 100.

Two readings follow. First, compression is refuted in the useful direction: the SSL view carries more text than the matched control (1,566 versus 1,179 tokens per candidate) and still scores higher at

Contrast	w=20	w=50	w=100
ssl − raw	+2.52	+1.29	+0.02
ssl − raw_head2048	+2.00	+0.39	+0.50
ssl − raw_match_ssl	+1.82	+0.39	+0.66
schedule − raw_head512	+3.71	+0.96	+0.35

Table 10: Paired nDCG@10 differences (pp) on the hard tier. On the easy tier at window 20: ssl − raw +3.06, ssl − raw_head2048 +3.17, ssl − raw_match_ssl +2.85, schedule − raw_head512 +3.15.

window 20. Second, at matched budget the advantage narrows, and none of the 24 paired differences survives correction for multiple comparisons.

For reference, the direct stage-1 orderings on the main run score 0.385 (raw) and 0.374 (SSL) at window 20 on the hard tier. The representation effect in Table 1 is measured between the reranked conditions.

E.2 Field and Label Ablations

To localize the scheduling-layer gain, we build ablation and control arms around the scheduling card and score each on a dense-retrieval endpoint. The ablation arms keep the identity prefix (name and description) of the view they modify and inherit its token budget. The component arms test the graph layer: sched_rel appends the scene–logic relations to the card, sched_facts appends the per-action facts, and rel_only keeps the relations alone without the card. The field arms strip the card: sched_no_intent removes the intent signature, while goal_only and goal_io keep only the skill goal, or the goal plus the expected inputs and outputs. The control arms hold length or rewriting constant: raw_match_schedule truncates the raw document to the card's token count, and the summ arms replace the card with a plain-prose condensation—full-length (summ_long), short (summ_short), or truncated to the card length (summ_match_short). The condensations are generated by the same backbone as the SSL records (DeepSeek-V4.1-Flash) from the complete source document, without SSL field labels or structure; Table 11 records each arm's budget.

View	Construction	Budget
raw_match_schedule	raw truncated per candidate to that candidate's card length	per-cand.
sched_rel	identity + scheduling card + scene–logic relations (no facts)	2,048
sched_facts	identity + scheduling card + per-action facts (no relations)	2,048
rel_only	identity + relations only	2,048
sched_no_intent	identity + goal, inputs/outputs, tags, dependencies (card minus intent)	512
goal_only	identity + skill goal	512
goal_io	identity + goal + expected inputs/outputs	512
summ_long	plain-prose condensation of the full document (median 1,419 tokens)	2,048
summ_short	purpose-written short condensation (median 212 tokens)	512
summ_match_short	summ_short truncated per candidate to the card length	per-cand.

Table 11: Construction of the ablation and summary-control views. All views embed the same candidate pool (79,141 hard-tier skills) with the same encoder.

The endpoint is the gold-rank percentile: for each task, the mean percentile of its core gold items in a view's stage-1 ordering of all candidates, where lower is better. Each row of Table 12 reports the paired percentile difference (first arm minus second arm; negative favors the first arm). The SkillRouter-Embedding-0.6B encoder runs the full grid; Qwen3-Embedding-0.6B (Zhang et al., 2025) re-encodes a fixed set of views to test encoder robustness: the scheduling card, the full SSL view, the flat card, the matched-length prose, the relations and facts arms, and raw. A contrast carries a Qwen3 column only when both of its views are in that set. The protocols pre-register two decision rules. First, a component carries measurable signal only if the arm that keeps it beats the arm built without it. Second, if the plain-prose condensation matches the scheduling card at window 20, the gain is attributable to LLM summarisation alone, and we would withdraw the claim that SSL's structure drives it.

Comparison (first − second)	SkillRouter-Emb.		Qwen3-Emb.†
	Δ	p	Δ, p
card − matched-length prose	−0.62	0.003	−0.46, 0.034†
card − short prose	−0.37	0.006	—
card − length-matched short prose	−0.20	0.067	—
labels removed (flat card − card)	+0.10	0.032	+0.22, 0.161†
+scene–logic relations (rel − card)	+0.42	0.0002	+0.20, 0.161†
+per-action facts (facts − card)	+0.40	0.0001	+0.13, 0.0098†
intent removed (no-intent − card)	−0.07	0.055	—
goal-only − card	−0.00	0.962	—
card − raw	−0.24	0.146	−0.57, 0.056†
full SSL − matched-length prose	−0.22	0.147	−0.29, 0.056†
full SSL − raw	+0.17	0.124	—

Table 12: Field and label ablations on the gold-rank percentile endpoint (hard tier, 200 core gold items; percentile difference in pp, lower is better, so negative favors the first view). The endpoint was chosen post hoc and all cells are exploratory. †Qwen3-Embedding-0.6B re-encodes the same views; rows without the dagger use a view outside that set, so they have no Qwen3 cell.

Three patterns match the main text. The scheduling card beats plain-prose condensations of the same document at matched and larger lengths, and this advantage replicates on the second encoder. Appending graph material to the card hurts on both encoders (relations and facts), which is why the full SSL view dilutes the card. The label effect (typed field names versus the same values unlabeled) is positive on the SkillRouter encoder but does not replicate on Qwen, so we do not claim it.

E.3 Serialization Examples

The three routing views of the same skill (`composable-rust-architecture`) illustrate what the retriever consumes. The scheduling card carries the typed invocation surface:

```
Name: composable-rust-architecture
Description: Expert knowledge for building event-driven systems with Composable
Rust framework. Use when implementing reducers, designing state machines, ...
Use for: Expert knowledge for building event-driven systems using the Composable
Rust framework - core architectural patterns, reducer design, effect
composition, and the unidirectional data flow model.
Intent examples: composable rust architecture
Intent examples: event-driven systems
Intent examples: reducer design
Intent examples: state machines
Intent examples: effect system
Intent examples: dependency injection
Intent examples: unidirectional data flow
...
```

The complete SSL view adds the structural and logical layers with typed labels; an excerpt:

```
Scene entry: design_feature (REASON: Design a new Composable Rust feature by
answering the Architecture Decision Checklist for State, Actions, Reducer,
Effects, and Environment.) -> design_state (select: State: What data needs to
be tracked? (Make it Clone, Serialize))
Scene transition: design_feature (...) -> verify_testing (VERIFY: Verify the
reducer can be tested without I/O using mocks.) when State, Actions, Reducer,
Effects, and Environment are identified
Scene contains step: design_feature (...) -> design_reducer (infer: Reducer:
```

```
What are the state transitions? (Pure function, fast))
Step transition: design_reducer (...) -> design_effects (select: Effects: ...)
when Reducer state transitions identified
Step transition: validate_testing (...) -> YIELD_SUCCESS (success) when Reducer
can be tested without I/O
Core operations: select: Effects: What side effects are needed? (Database, HTTP,
events, delays). Effects produce new Actions: ...
Critical conditions: Reducer can be tested without I/O
...
```

The flat view renders the identical values with field labels removed, which is the control used in the label ablation above.

F Earlier Delivery Tries for Execution

Before the action card, two simpler delivery tries were run on the same SkillsBench tasks and produced no reliable benefit; they motivate the card's design.

First, the SSL record itself as prompt material: appending the selected skills' SSL text to the agent's prompt leaves verified success unchanged (paired against raw: -0.70 pp on DeepSeek-V4.1-Flash and $+2.00$ pp on GLM-5.3-Flash).

Second, a terminal semantic review turn: after the agent's first turn, the harness sends one review message, in three variants (a plain prompt, the same prompt with the flat passage list, and the same prompt with source-grounded check lines). The mechanism works—the review turn completes in most cells and issues real mutating tool calls—but none of the variants moves verified success (e.g. $+1.76$ pp for the source-grounded variant against raw on 152 runs).

G Skill Execution: Card Construction and Conditions

Cards are generated deterministically from the skill files of each task. A task is eligible when at least two of its logic steps ground quotes in those files and one card pair fits the fixed construction parameters, which covers 76 of the 87 SkillsBench tasks. The annotations come from the record's logical-layer steps (act_type, object, instrument, input_args, output_binding, preconditions, effects), and the quoted passages are the source spans that ground these fields. Each card lists selected passages from the task's skill files word for word, as `path:lines` entries with the quoted source text, in source order. The `card_ssl` variant adds one annotation line per passage, of the form input/object $\rightarrow$ instrument [result: binding], plus a trailing `check:` line naming the expected verification. Two checks on the generated text report no violations: the text matches the source character for character, and every quoted span matches the source byte-for-byte. Regenerating the assets reproduces them identically. Each card is built under the fixed parameters listed in Table 13.

Parameter	**Value**
Card length cap	1,800 characters
Length parity between `card_ssl` and `card_flat`	15%
Quoted-window floor	40 characters
Quote cap per entry	220 (`card_ssl`), 900 (`card_flat`)
Entries per card	2–4
Skill files quoted per card	at most 2
Check-line quote	prose, target 90 characters, cap 220
Eligibility	≥ 2 grounded steps and a fitting pair

Table 13: Card construction parameters. The released code exposes the same constants.

```
Action card ada-bathroom-plan-repair | sources:
  skills/architectural-dxf-extraction/SKILL.md,
  skills/geometric-layout-repair/SKILL.md
- candidate_repaired_layout -> Shapely [result: validated_repaired_layout]
  source: skills/geometric-layout-repair/SKILL.md:55-56 "- Validate polygons
  with Shapely before writing JSON. - Ensure all fixture bounding boxes are
  covered by the room polygon."
  check: "- Validate polygons with Shapely before writing JSON. - Ensure all
  fixture bounding boxes are covered by the room polygon."
```

The excerpt above is one `card_ssl` card with line breaks added for display. The flat variant carries the same quoted spans without the annotation and `check:` lines.

The two ablation arms reuse the card pipeline with one alteration each. `card_ssl_rand` delivers the `card_ssl` card of a different task, so the annotations are well-formed but unrelated to the task at hand. `card_ssl_shuffle` delivers the task's own card with the annotation lines permuted across passages, so each claim is paired with the wrong evidence. Both arms come from a separate run in which `raw`, `card_flat`, and `card_ssl` are re-run under the same protocol, so every contrast involving the ablation arms is computed within that run.

Card arms also read the task's own skill files more often than raw: mean original-skill reads per run are 3.08 (`card_ssl`) and 2.91 (`card_flat`) against 2.43 (`raw`) on DeepSeek-V4.1-Flash, and 1.11 and 1.33 against 0.85 on GLM-5.3-Flash.

H Reproducibility, Release, and Responsible Use

All reported experiments are inference-only: reproducibility depends on hosted-model inference, embedding generation, ranking, and aggregation; no model training or fine-tuning is involved. The routing pool is one normalization pass over 79,141 candidate skills (Appendix D), and the action-card pipeline is deterministic, so the remaining inference sits in the agent runs themselves.

The main reproducibility cost is LLM inference, while local computation is comparatively lightweight. Routing requires about 79,000 calls plus the challenge-set reruns and the single regenerated record documented in Appendix D; the retrieval, reranking, and bootstrap analyses run locally over the normalized records. Execution requires the agent runs: five card conditions on 76 eligible tasks for each of two agent models, and the earlier delivery channels reported in Appendix F; card generation is deterministic. A full rebuild is therefore dominated by hosted-model latency.

The released artifacts are the SSL schema, the normalizer prompt, and the minimal supporting code for record validation, view serialization, and action-card construction; normalized records and the per-run artifacts behind the reported tables are not part of the release. The SkillRouter Benchmark and SkillsBench are external and remain subject to their own licenses and attribution conditions.

I Case Studies

We read one case per evaluation from the recorded runs to make the aggregate trends concrete.

I.1 Skill Routing: Two Skills Recovered at a Window of Twenty

In the hard tier, the task `energy-ac-optimal-power-flow` asks the agent to verify that a day-ahead schedule is AC-feasible and to report an operating point, and the benchmark labels three skills as gold: `ac-branch-pi-model`, `casadi-ipopt-nlp`, and `power-flow-data`. Under the raw view, no gold skill reaches the top-20 of the 79,141 candidates, so the reranked window of twenty yields $nDCG@10 = 0.00$. Under the SSL view, the same encoder and the same window place `ac-branch-pi-model` at rank 7 and `power-flow-data` at rank 16, the reranker puts a gold skill first, and window-20 nDCG@10 rises to 0.60. The matched records state the invocation surface directly: `ac-branch-pi-model` lists intents such as "compute AC branch pi-model power flows" and "handle transformer tap ratio and phase shift", and `power-flow-data` lists "parse bus, generator, and branch

data" and "MATPOWER network data". The third gold skill, `casadi-ipopt-nlp`, moves from rank 24 to rank 57, so the SSL view reroutes the ranking instead of promoting every gold skill.

I.2 Skill Execution: A Quoted Example Versus a Named Decision

`simpo-code-reproduction` asks the agent to reproduce a training repository and to record its Python environment, and the verifier accepts only Python 3.10. The task's skill tells the agent to compare the repository's declared Python version against the current environment and to set up a matching one, and the same section carries an illustrative snippet that sets up Python 3.11. The flat card quotes that section with the example included; the annotated card binds the step as `dependency_declarations, current_environment_log -> [result: python_version_match_decision]`, and its quoted span stops before the example. On DeepSeek-V4.1-Flash, the raw run and the flat-card run both install Python 3.11.8, and the verifier records that the environment information does not contain Python 3.10, while the annotated card installs Python 3.10.14 and passes; the three rewards are 0.0, 0.0, and 1.0. On GLM-5.3-Flash the raw run fails in the same way and both card conditions pass, so the annotation contrast on this task is DeepSeek-specific. The case isolates what the annotations carry: the flat card transmits the skill's example, and the annotated card transmits the skill's decision.