TRACE: TASK-CONDITIONED RETRIEVAL AND AGGREGATION WITH TOKEN-BUDGETED COMPILATION FOR EXECUTABLE SKILL PROGRAMS IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Skills provide a scalable abstraction for LLM agents by encapsulating reusable procedures, tool affordances, implementation details, validation logic, and failure-handling strategies. However, effectively leveraging large-scale skill repositories remains challenging: existing retrieval methods mainly optimize individual skill relevance, while existing compression methods often reduce context without preserving the execution structure required for multi-skill tasks. We propose TRACE, a task-conditioned skill retrieval, aggregation, and compilation framework that bridges the gap between skill retrieval and reliable execution. TRACE first constructs complementary skill bundles through requirement-conditioned graph routing with dynamic edge activation and then performs token-budgeted compilation to transform retrieved skills into compact executable programs containing dependencies, operations, validators, and fallback policies. For failed executions, TRACE further applies trace-guided repair to patch task-specific executable programs using execution traces and failure feedback without modifying the persistent skill library. Experiments on 51 multi-skill tasks show that TRACE achieves 92.16% execution success with only 2.18K estimated skill-input tokens on average, reducing skill-input cost by approximately 90% compared with raw top-20 skill injection. Moreover, trace-guided repair recovers two of four failed executions and improves the final success rate to 96.08%. Ablation studies further demonstrate the importance of role-balanced selection, validation information, and fallback guidance for reliable skill execution.

1 INTRODUCTION

Skills are becoming an important abstraction for building capable LLM agents. Recent agent systems enable language models to use tools, invoke APIs, and perform complex tasks through reusable capabilities (Schick et al., 2023; Yao et al., 2023; Qin et al., 2023; Patil et al., 2023; Shen et al., 2023; Wang et al., 2023). A skill encapsulates task-specific procedures, tool affordances, implementation details, validation logic, and failure-handling strategies into a reusable unit. However, exposing large skill repositories directly to agents introduces substantial context overhead and irrelevant information interference, making exhaustive skill inspection infeasible as repositories scale. Therefore, effective agents must first retrieve relevant skills and transform them into compact executable representations for downstream execution.

The first challenge is effective skill retrieval. Existing retrieval systems can be categorized into lexical retrieval, dense retrieval, interaction-based retrieval, skill-oriented routing, and structure-aware retrieval. Lexical methods such as BM25 (Robertson and Zaragoza, 2009) rely on term matching, while dense retrievers such as DPR (Karpukhin et al., 2020) improve semantic matching through learned representations. Interaction-based methods such as ColBERT (Khattab and Zaharia, 2020) and cross-encoder rerankers (Nogueira and Cho, 2019) further enhance ranking precision through fine-grained query-skill interactions. Recent skill-oriented methods such as SkillRouter (Zheng et al., 2026) demonstrate the importance of skill-aware routing at scale. However, these methods mainly optimize individual skill relevance, which is insufficient for multi-skill tasks requiring coordinated capabilities across prerequisites, operations, validators, and fallback procedures. Structure-aware retrieval methods such as GraphRAG (Edge et al., 2024) and HippoRAG (Gutierrez et al.,

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.

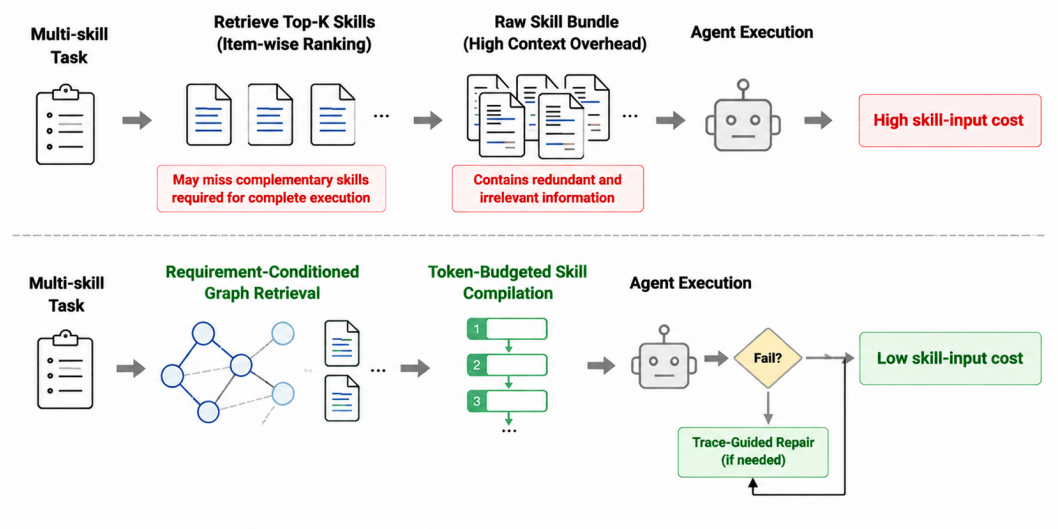

Figure 1: Motivation of the retrieval-to-execution problem. Direct item-wise skill retrieval can miss complementary skills and pass a redundant raw skill bundle to the agent, making downstream execution costly and failure-prone. Our framework retrieves requirement-supported skill bundles, rewrites them into compact executable programs, and applies trace-guided repair only when needed.

2024) model relationships among items, but unrestricted expansion may introduce functionally irrelevant skills. Therefore, multi-skill retrieval requires requirement-aware construction of complementary skill bundles rather than independent skill ranking.

The second challenge is effective skill compilation. Existing approaches include context compression, retrieval-aware compression, skill-aware compilation, and execution-feedback-based refinement. Context compression methods such as LLMLingua (Jiang et al., 2023a) reduce input length by removing less important information, while RECOMP (Xu et al., 2023) performs retrieval-aware compression by distilling useful retrieved contexts. Recent skill-oriented methods such as SkillRAE (Meng et al., 2026) further explore skill-specific context compilation. However, these approaches primarily focus on information reduction or context selection rather than constructing execution-oriented representations. In multi-skill settings, naive compression may remove critical execution information, including dependencies, operation ordering, validation conditions, and recovery strategies. Therefore, effective skill compilation requires transforming retrieved skills into structured executable representations that preserve actions, constraints, and failure-handling procedures.

To address these challenges, we propose TRACE, a token-budgeted skill compilation framework for multi-skill LLM agents. TRACE first performs requirement-conditioned graph routing with dynamic edge activation to retrieve a high-coverage skill bundle. It then compiles retrieved skills into compact executable programs by organizing execution-critical information into dependencies, actions, validators, and fallbacks. Finally, TRACE applies trace-guided repair to recover failed executions using execution traces and failure outputs without modifying the persistent skill library.To sum up, our contributions are:

- We propose TRACE, a token-budgeted skill compilation framework for multi-skill LLM agents that integrates requirement-conditioned graph retrieval and executable program generation for reliable skill utilization.
- We introduce a requirement-conditioned skill retrieval and token-budgeted compilation mechanism that constructs complementary skill bundles and transforms retrieved skills into compact executable programs with explicit dependencies, actions, validators, and fallback policies. We further propose trace-guided repair, a conditional program repair mechanism that patches task-specific executable programs using execution traces and failure outputs without modifying the persistent skill library.

- We conduct extensive experiments on 51 multi-skill tasks. The results show that TRACE reduces estimated skill-input tokens by approximately 90% while maintaining competitive execution performance, and trace-guided repair further improves the final success rate.

2 Related Work

2.1 Skill Retrieval and Selection for LLM Agents

Retrieving appropriate skills from large repositories is a fundamental problem for skill-based LLM agents. Existing retrieval approaches can be broadly categorized into lexical retrieval, dense semantic retrieval, and structure-aware retrieval. Lexical retrieval methods identify relevant skills through explicit term matching. BM25 (Robertson and Zaragoza, 2009) is a representative sparse retrieval method that ranks candidates based on term frequency and inverse document frequency, providing effective matching when queries and skills share similar vocabulary. Dense retrieval methods instead learn semantic representations to capture implicit relationships between queries and candidates. DPR (Karpukhin et al., 2020) improves retrieval through dense query and passage representations, while Sentence-BERT (Reimers and Gurevych, 2019) and ANCE (Xiong et al., 2021) further enhance semantic matching through learned embedding spaces and contrastive training. Interaction-based retrieval methods improve ranking precision by modeling fine-grained query-candidate interactions. ColBERT (Khattab and Zaharia, 2020) introduces efficient late interaction through token-level similarity computation, while cross-encoder reranking methods (Nogueira and Cho, 2019) improve precision through joint query-candidate encoding. These approaches have substantially improved the ability of LLM agents to locate relevant external knowledge and capabilities, but they mainly optimize relevance estimation for individual retrieval candidates. Multi-skill tasks often require coordinated sets of complementary skills containing prerequisites, operational procedures, validation mechanisms, and fallback strategies, where independently ranking skills may retrieve highly related skills while missing supporting capabilities required for complete execution. Recent skill-oriented routing methods such as SkillRouter (Zheng et al., 2026) investigate large-scale skill selection by leveraging skill information for routing decisions, but they still focus on selecting individual skills rather than constructing requirement-aware skill bundles. Structure-aware retrieval methods provide another direction by modeling relationships among retrieved items. GraphRAG (Edge et al., 2024), RAPTOR (Sarthi et al., 2024), and HippoRAG (Gutierrez et al., 2024) explore graph-based or hierarchical retrieval to improve multi-hop information access and retrieval coverage. However, unrestricted graph expansion may introduce semantically related but functionally irrelevant skills. Therefore, existing retrieval methods remain insufficient for multi-skill execution, motivating approaches that construct high-coverage skill bundles according to task requirements.

2.2 Skill Compilation and Execution Feedback for LLM Agents

After retrieval, retrieved skills need to be transformed into representations suitable for reliable execution. Existing approaches can be broadly categorized into context compression, skill-aware compilation, and execution-feedback-based refinement. Context compression methods aim to reduce input length by removing redundant or less informative information. Selective Context (Li et al., 2023b) identifies and preserves important context segments, while LLMLingua (Jiang et al., 2023a) and LongLLMLingua (Jiang et al., 2023b) compress prompts through token-level and structure-aware reduction. RECOMP (Xu et al., 2023) further explores retrieval-aware compression by distilling useful information from retrieved contexts. These methods effectively reduce inference cost, but they mainly optimize information reduction rather than constructing structured execution procedures, and relevance-based compression may remove execution-critical information such as required inputs, operation ordering, validation conditions, and recovery strategies. Recent approaches investigate skill-aware compilation and structured reasoning. SkillRAE (Meng et al., 2026) studies skill-specific context compilation by selecting relevant skill subunits and reducing unnecessary information before execution. Program-aided reasoning methods such as PAL (Gao et al., 2023) and Program-of-Thoughts prompting (Chen et al., 2022) introduce intermediate executable representations to improve reasoning reliability. However, these approaches mainly generate compact contexts or reasoning structures rather than executable skill representations that explicitly encode dependencies, actions, validators, and fallback policies. Another line of work improves agent reliability through execution feedback and iterative refinement. Reflexion (Shinn et al., 2023) and Self-Refine

(Madaan et al., 2023) use feedback signals to improve subsequent responses, while software engineering benchmarks and agents such as SWE-bench (Jimenez et al., 2024) and SWE-agent (Yang et al., 2024) leverage execution outcomes for error correction. Nevertheless, these methods primarily refine reasoning trajectories or reusable behaviors rather than repairing task-specific executable representations generated from retrieved skills. Therefore, existing approaches cannot directly transform retrieved skill bundles into compact executable programs while preserving execution-critical information and supporting targeted repair after failures, motivating skill compilation methods that explicitly model execution logic and failure handling.

3 Problem Formulation

We formulate multi-skill execution for LLM agents as a skill-based function calling problem. Given a task query q and a skill repository $S = \{s_1, \ldots, s_N\}$, where N is the total number of available skills, the objective is to construct an executable program that enables the agent to complete the task. Each skill s_i contains an executable capability together with associated procedures, requirements, validation rules, and fallback strategies.

The retrieval stage selects a task-relevant skill bundle:

$$B_K(q) = \mathrm{Retrieve}(q, S),$$

where $B_K(q) \subseteq S$ denotes the retrieved top-K skills. The compilation stage transforms the retrieved skills into an execution-oriented program:

$$P(q) = \mathrm{Compile}(q, B_K(q)),$$

where $P(q)$ preserves execution-critical information, including procedures, dependencies, validation conditions, and fallback strategies. The downstream agent executes:

$$y = \mathcal{A}(q, P(q), \mathcal{E}),$$

where $\mathcal{E}$ denotes the execution environment. The goal is to construct compact executable programs that maintain execution reliability while reducing unnecessary context.

4 Methodology

TRACE is a token-budgeted skill compilation framework for multi-skill LLM agents. It addresses multi-skill execution through three stages: requirement-conditioned graph routing, token-budgeted skill compilation, and trace-guided repair. The first stage constructs a complementary skill bundle by activating requirement-supported relations in a skill graph. The second stage transforms retrieved skills into compact executable programs by preserving execution-critical information. The final stage applies trace-guided repair to update failed programs using execution feedback without modifying the persistent skill library.

4.1 Requirement-Conditioned Graph Retrieval

Given a task query q and a large skill repository S, the retrieval module outputs a high-coverage top-20 skill bundle $B_K(q)$. Unlike conventional retrieval methods that optimize individual skill relevance, TRACE aims to identify complementary skills required for complete task execution.First, a requirement compiler extracts task requirements, including required artifacts, operations, prerequisites, outputs, constraints, and verification conditions. These requirements are used to retrieve initial candidates through dense similarity, lexical matching, and requirement-aware signals. Dense retrieval anchors are preserved to maintain reliable candidates.Next, TRACE constructs skill bundles through requirement-conditioned graph routing. An offline skill graph provides dependency, workflow, complement, and conflict relations among skills. Instead of activating all neighboring skills, TRACE only activates graph edges when the connected skills are supported by extracted task requirements. This prevents graph expansion from introducing semantically related but functionally irrelevant skills.Finally, candidates are reranked using requirement-aware evidence aggregation that combines semantic similarity, lexical evidence, requirement support, graph support, anchor preservation, and conflict penalties. The output of this stage is a top-20 skill bundle consumed by the downstream compilation module.

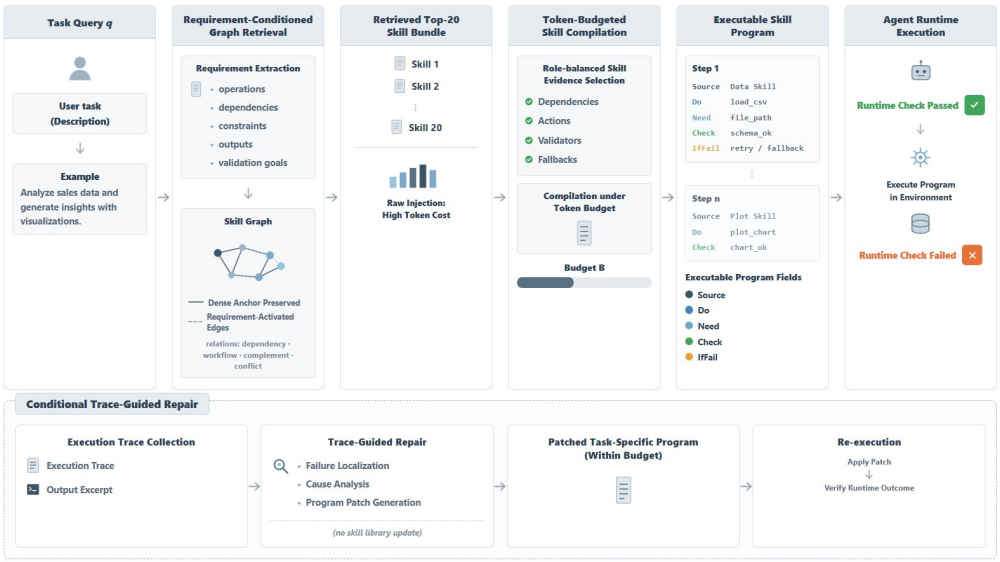

Figure 2: Overview of the TRACE retrieval–compilation–repair pipeline. TRACE first retrieves a requirement-conditioned top-20 skill bundle, then compiles the retrieved skills into a token-budgeted executable program. If the program fails during downstream execution, trace-guided repair patches the task-specific program without modifying the skill library.

4.2 Token-Budgeted Skill Compilation

Given the retrieved skill bundle $B_K(q)$, TRACE transforms raw skill descriptions into a compact executable program. The compilation process extracts candidate evidence fragments from skill names, descriptions, and bodies, and assigns each fragment a functional role:

$$r \in \{\text{dependency}, \text{action}, \text{validator}, \text{fallback}, \text{support}\}.$$

The role assignment identifies different types of execution information, where dependencies describe required conditions, actions specify operations, validators define correctness checks, fallbacks provide recovery strategies, and support evidence contains additional context. TRACE removes duplicate or redundant fragments and selects useful evidence under a configurable token budget based on task relevance and role coverage.The selected fragments are rewritten into an executable program rather than a simple summary of retrieved skills. The program organizes information into execution goals, preconditions, ordered steps, validation checks, and fallback policies. Each execution step is represented as:

$$\text{Step} = (\text{Source}, \text{Do}, \text{Need}, \text{Check}, \text{IfFail}),$$

where Source records the originating skill, Do specifies the operation, Need lists required inputs or dependencies, Check defines validation conditions, and IfFail provides recovery actions. This structured representation preserves execution-critical information while reducing unnecessary context.

4.3 Trace-Guided Repair

Trace-guided repair handles cases where the compiled program fails during downstream execution. Instead of retrieving new skills or modifying the persistent skill repository, TRACE performs local repair on the task-specific executable program.Given a failed execution, the repair module receives the original program together with execution traces, error information, and output feedback. These signals identify missing details, incorrect assumptions, insufficient validation conditions, or incomplete fallback strategies. TRACE then patches only the problematic parts of the current program while preserving useful execution steps.This design keeps repair bounded to the failed execution instance and avoids introducing changes to the reusable skill library. Successful executable programs remain unchanged, while repair is applied only to initially failed executions to recover execution reliability.

5 EXPERIMENTS

5.1 SETTINGS

Dataset.

We evaluate on 51 multi-skill tasks derived from the SkillsBench-style setting. Each task is associated with a top-20 retrieved skill bundle. The retrieval experiments evaluate bundle quality under Easy and Hard candidate-pool settings, while downstream experiments evaluate how different agent-facing representations affect execution success and token cost. Downstream tasks are executed in a standard coding-agent framework with DeepSeek-V4-Flash (DeepSeek-AI, 2026) as the backend model.

Baselines.

Retrieval baselines. We compare TRACE retrieval against BM25 full-text retrieval, dense retrieval, full-text retrieve-and-rerank, SkillRouter (Zheng et al., 2026), and requirement-conditioned graph routing.

Downstream baselines. We compare four agent-facing representations constructed from the same retrieved top-20 skill evidence. **raw_top20** directly injects all top-20 skill bodies without applying any token budget. **SkillRAE** applies SkillRAE (Meng et al., 2026) for native skill-context compilation. **TRACE** denotes our token-budgeted skill compilation method. **TRACE**^{repair} retains successful TRACE^{base} executions and applies trace-guided repair only to tasks where TRACE^{base} initially fails.

Metrics.

Retrieval metrics. We evaluate skill retrieval using Recall@20 and Requirement Coverage@20. Recall@20 measures the fraction of gold skills recovered in the retrieved top-20 bundle:

$$\text{R@20} = \frac{|B_{20}(q) \cap G(q)|}{|G(q)|},$$

where $B_{20}(q)$ is the retrieved skill bundle and $G(q)$ is the set of gold skills for task q. Requirement Coverage@20 measures the fraction of extracted task requirements supported by at least one retrieved skill:

$$\text{RC@20} = \frac{|\{r \in R(q) : \exists s \in B_{20}(q), \text{termoverlap}(r, s) > 0\}|}{|R(q)|},$$

where $R(q)$ denotes the requirement cues extracted by `rule_compile_requirements`.

Execution metrics. For downstream execution, we report success rate, average skill-input tokens, average total tokens, and repair recovery rate. A run is considered successful when the coding-agent process exits with code 0 and contains none of the predefined failure markers. Token counts are estimated as:

$$\hat{T}(x) = \max(1, |\text{to_text}(x)|/4),$$

where $\text{to_text}(x)$ converts an input representation into text. For TRACE^{repair}, repair recovery rate measures the fraction of initially failed TRACE^{base} executions successfully recovered by trace-guided repair:

$$\text{RepairRecovery} = \frac{|\{i : y_i^{\text{base}} = 0, y_i^{\text{repair}} = 1\}|}{|\{i : y_i^{\text{base}} = 0\}|}.$$

5.2 COMPARISON

To evaluate TRACE from both retrieval and execution perspectives, we conduct two groups of comparisons. For skill retrieval, we compare requirement-conditioned graph routing with lexical, dense, reranking-based, and skill-routing baselines under Easy and Hard candidate-pool settings. Table 1 and Figure 3 report Recall@20 and Requirement Coverage@20. Requirement-conditioned graph routing improves Recall@20 over the retrieval baselines and achieves complete coverage of the extracted requirement cues under the term-overlap coverage metric. This suggests that dynamic edge activation helps retrieve complementary skills that provide textual support for task requirements,

Method	Easy R@20	Hard R@20	Avg R@20	Easy RC@20	Hard RC@20	Avg RC@20
BM25	0.562	0.538	0.551	0.609	0.586	0.598
Dense Retrieval	0.538	0.514	0.526	0.609	0.586	0.598
Retrieve+Rerank	0.645	0.627	0.636	0.686	0.680	0.683
SkillRouter	0.651	0.639	0.645	0.692	0.681	0.687
TRACE	0.657	0.645	0.651	1.000	1.000	1.000

Table 1: Retrieval comparison on 51 multi-skill tasks. R@20 measures partial gold-skill recovery, while RC@20 measures the fraction of task-derived requirement cues textually supported by the retrieved top-20 bundle.

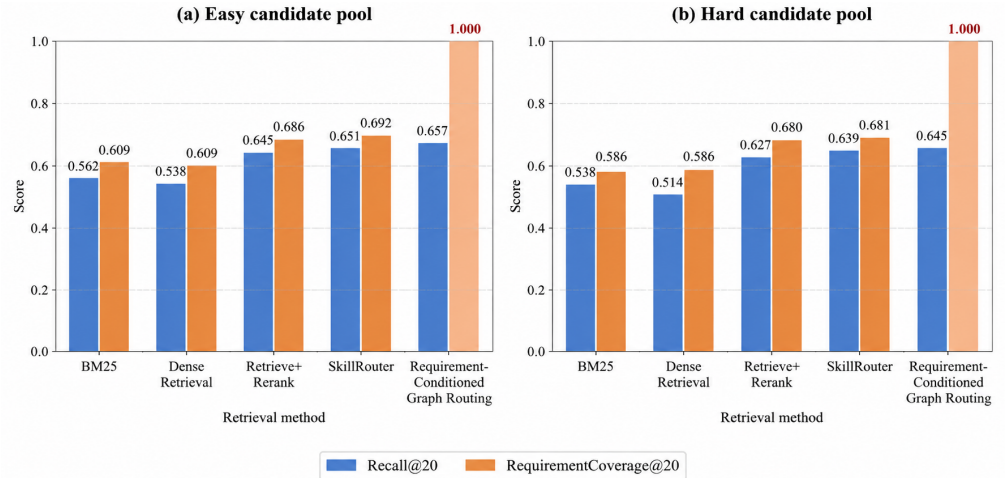

Figure 3: Retrieval performance under Easy and Hard candidate-pool settings. The figure compares Recall@20 and RequirementCoverage@20 across BM25, dense retrieval, retrieve-and-rerank, SkillRouter, and requirement-conditioned graph routing. Requirement-conditioned graph routing achieves complete requirement-evidence coverage under the term-overlap RC@20 metric while maintaining competitive skill recovery.

even when not all gold skills are recovered. The RC@20 metric measures requirement-evidence coverage rather than strict full gold-skill coverage.

Figure 3 visualizes the retrieval results under the Easy and Hard candidate-pool settings. Requirement-conditioned graph routing achieves complete requirement-evidence coverage in both settings while maintaining the highest Recall@20 among the compared methods. This indicates that requirement-conditioned graph activation improves requirement-evidence coverage under the reported term-overlap RC@20 metric without relying on unrestricted graph expansion.For skill compilation and downstream execution, Table 2 reports results on all 51 tasks under the 4096-token method-side construction budget with estimated token accounting. Raw top-20 and SkillRAE are not constrained by this budget.The downstream results show that raw top-20 achieves 94.12% success but requires 21.81K estimated skill-input tokens on average, confirming that full skill bodies provide useful execution information but introduce substantial context cost. SkillRAE reduces the average skill-input tokens to 3.30K, but its success rate decreases to 84.31%, suggesting that compact context alone may remove execution-critical structure. TRACE^{base} uses only 2.18K estimated skill-input tokens on average, reducing the estimated skill-input cost by approximately 90% compared with raw top-20 and by 33.8% compared with SkillRAE, while maintaining 92.16% success. TRACE^{repair} further recovers two of the four failed TRACE^{base} executions and improves the final success rate to 96.08%. This conditional result is computed from per-task logs: TRACE^{base} succeeds on 47 of 51 tasks, and trace-guided repair recovers two of the four initially failed executions, resulting in 49 successful executions overall. The TRACE^{repair} token estimate is computed using the same conditional accounting procedure, retaining TRACE^{base} cost for initially successful tasks and adding repair cost only for initially failed tasks.

Representation	Count	Success Rate	Avg Skill Tokens	Avg Total Tokens	Total Tokens
raw_top20	51	94.12	21805.63	22502.10	1147607
SkillRAE	51	84.31	3300.90	4123.78	210313
$\text{TRACE}^{\text{base}}$	51	92.16	2184.67	2905.94	148203
$\text{TRACE}^{\text{repair}}$	51	96.08	2184.67	3062.14	156169

Table 2: Downstream tasks are executed in a standard coding-agent framework with DeepSeek-V4-Flash (DeepSeek-AI, 2026) as the backend model. Raw top-20 and SkillRAE are unconstrained baselines. $\text{TRACE}^{\text{base}}$ uses a 4096-token method-side construction budget with estimated token accounting. $\text{TRACE}^{\text{repair}}$ is a conditional metric computed from per-task logs: initially successful $\text{TRACE}^{\text{base}}$ runs are retained, and repair outcomes and costs are counted only for tasks whose original $\text{TRACE}^{\text{base}}$ run failed.

	$\text{TRACE}^{\text{base}}$		$\text{TRACE}^{\text{repair}}$	
Variant	Success Rate	Avg Skill Tokens	Success Rate	Avg Skill Tokens
Full method	92.16	2184.67	96.08	2184.67
w/o validators	45.10	2184.67	68.63	2184.67
w/o fallbacks	68.63	2060.71	78.43	2060.71
w/o role buckets	3.92	2006.41	3.92	2006.41

Table 3: Component ablation study on 51 multi-skill tasks. The $\text{TRACE}^{\text{base}}$ columns report success before repair. The $\text{TRACE}^{\text{repair}}$ columns are conditional metrics that retain successful $\text{TRACE}^{\text{base}}$ runs and count repair outcomes only for tasks that initially fail. In the fallback ablation, fallback-role fragments and the global fallback policy are removed, while the generic step-level failure instruction from the rendering template remains.

5.3 Ablation Study

We conduct component ablations to examine whether the structure of the executable program contributes to downstream reliability beyond simple context compression. Each variant removes one component while keeping the retrieved top-20 bundle, method budget, downstream agent, and execution protocol unchanged. **w/o validators** removes validator fragments, step-level checks, and global validation instructions from the executable program. **w/o fallbacks** removes fallback-role fragments and the global fallback policy while retaining the generic step-level failure instruction generated by the current rendering template. **w/o role buckets** selects fragments globally by task relevance without role-specific selection quotas.The conditional repair outcomes are computed from per-task execution logs by retaining initially successful $\text{TRACE}^{\text{base}}$ executions and applying repair only to failed cases. This decomposition distinguishes the direct contribution of executable-program construction from the additional contribution of trace-aware repair.

The ablation results show that TRACE produces a structured execution artifact rather than a simple compressed summary. For the full method, $\text{TRACE}^{\text{base}}$ directly succeeds on 47 of 51 tasks, and trace-guided repair recovers 2 of the remaining 4 failed executions, resulting in 49 successful executions overall. Removing role buckets causes the largest degradation: executable-program success falls from 92.16% to 3.92%, and conditional repair remains at 3.92%. This indicates that globally ranking fragments by relevance is insufficient in the current rendering procedure, because the resulting context can lose the balance among dependencies, actions, validators, and fallback-related evidence required for execution. Removing validators reduces executable-program success to 45.10%. trace-guided repair improves this setting to 68.63%, but it does not recover the performance of the full method, suggesting that trace-conditioned repair cannot fully compensate for validation information removed during compilation. This confirms that explicit validation information is important for checking intermediate artifacts, output formats, and task constraints. Removing fallback-role fragments and the global fallback policy reduces executable-program success to 68.63%. trace-guided repair improves this setting to 78.43%, but still remains below the full method. The decrease indicates that explicit recovery guidance is useful when operations fail or preferred routes are un-

available. Because the current rendering template still emits a generic step-level IFFAIL instruction, this ablation should be interpreted as removing fallback evidence and the global fallback policy rather than removing all failure-handling information. Overall, the ablations support a complementary interpretation of the executable program: role buckets provide structural balance, validators support correctness, and fallback-related guidance improves recoverability.

5.4 CASE STUDY

We analyze one representative vulnerability-repair task to illustrate how different skill representations affect downstream execution.

CVE-oriented vulnerability repair. The task *fix-druid-loophole-cve* requires the agent to produce a concrete fix for a vulnerability-related build or migration problem. Table 4 summarizes the execution outcomes under different skill representations.

Representation	Execution	Skill-input Tokens
raw_top20	Fail	–
SkillRAE	Success	367
TRACE executable program	Success	557

Table 4: Case study on the CVE-oriented vulnerability repair task. Token counts are estimated using the same accounting procedure as the main experiments.

As shown in Table 4, the raw top-20 skill bundle fails on this task, while both SkillRAE and TRACE succeed with compact skill inputs. TRACE transforms retrieved skills into an executable program containing task-relevant execution information rather than directly exposing the unstructured skill bundle.

This case suggests that structured skill representations can provide useful execution guidance for vulnerability-repair tasks. However, this single example does not isolate the effect of context length from other factors, including prompt organization, execution variability, or run-level success criteria.

6 CONCLUSION

Skill-based agents provide a promising paradigm for scaling LLM agents by enabling reusable capabilities and modular task execution. However, effectively utilizing large-scale skill repositories remains challenging, as existing methods mainly focus on retrieving relevant skills or compressing contexts, while often ignoring skill composition, execution structure preservation, and failure recovery.In this work, we present TRACE, a token-budgeted skill compilation framework that addresses these challenges through requirement-conditioned graph routing, executable program generation, and trace-guided repair. TRACE constructs complementary skill bundles, compiles retrieved skills into structured executable programs with dependencies, operations, validators, and fallback policies, and performs task-specific repair based on execution feedback without modifying the skill library. Experiments on 51 multi-skill tasks show that TRACE reduces skill-input cost while maintaining reliable execution performance, and ablation studies demonstrate the importance of structured skill selection and execution guidance.As skill-based LLM agents are still an emerging research direction, future work will explore more scalable skill organization, adaptive compilation strategies, and continuous improvement mechanisms based on long-term execution feedback.

AI USE STATEMENT

We used generative AI tools to assist with polishing the writing, improving clarity, and refining the presentation of the manuscript. We did not use generative AI tools for research ideation, experiment execution, data generation, mathematical proof, or drafting sections of the paper.

All AI-assisted suggestions were reviewed and revised by the authors. The authors verified the correctness of the technical content, experimental results, and claims presented in this work, and take full responsibility for the final content of the manuscript, including all text, analyses, and artifacts.

Ethics Statement

This work studies methods for improving LLM agent skill retrieval, execution, and reliability. Potential concerns include misuse of autonomous agents and deployment in sensitive environments. The proposed framework does not modify persistent skill libraries automatically and restricts repair operations to task-specific execution programs. We encourage responsible deployment, evaluation under appropriate safeguards, and careful consideration of downstream applications.

Reproducibility Statement

We provide detailed descriptions of the retrieval, aggregation, rewriting, and repair procedures in the main paper. The experimental settings, evaluation metrics, token accounting procedure, and ablation configurations are described to support reproducibility. The released implementation should include the downstream context construction, execution, repair, and summarization scripts, together with the precomputed retrieval outputs used by the downstream experiments.

References

DeepSeek-AI. 2026. DeepSeek-V4: Towards Highly Efficient Million-Token Context Language Models. *arXiv preprint arXiv:2606.19348*.

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; Herzog, A.; Ho, D.; Hsu, J.; Ibarz, J.; Ichter, B.; Irpan, A.; Jang, E.; Ruano, R. J.; Jeffrey, K.; Jesmonth, S.; Joshi, N.; Julian, R.; Kalashnikov, D.; Kuang, Y.; Lee, K.-H.; Levine, S.; Lu, Y.; Luu, L.; Parada, C.; Pastor, P.; Quiambao, J.; Rao, K.; Rettinghouse, J.; Reyes, D.; Sermanet, P.; Sievers, N.; Tan, C.; Toshev, A.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Xu, S.; Yan, M.; and Zeng, A. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of CoRL*.

Asai, A.; Wu, Z.; Wang, Y.; Sil, A.; and Hajishirzi, H. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *Proceedings of ICLR*.

Borgeaud, S.; Mensch, A.; Hoffmann, J.; Cai, T.; Rutherford, E.; Millican, K.; van den Driessche, G.; Lespiau, J.-B.; Damoc, B.; Clark, A.; de Las Casas, D.; Guy, A.; Menick, J.; Ring, R.; Hennigan, T.; Huang, S.; Maggiore, L.; Jones, C.; Cassirer, A.; Brock, A.; Paganini, M.; Irving, G.; Vinyals, O.; Osindero, S.; Simonyan, K.; Rae, J.; Elsen, E.; and Sifre, L. 2022. Improving Language Models by Retrieving from Trillions of Tokens. In *Proceedings of ICML*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*.

Chen, W.; Ma, X.; Wang, X.; and Cohen, W. W. 2022. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *arXiv preprint arXiv:2211.12588*.

Chen, X.; Lin, M.; Schärli, N.; and Zhou, D. 2023. Teaching Large Language Models to Self-Debug. *arXiv preprint arXiv:2304.05128*.

Edge, D.; Trinh, H.; Cheng, N.; Bradley, J.; Chao, A.; Mody, A.; Truitt, S.; and Larson, J. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130*.

Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; and Neubig, G. 2023. PAL: Program-aided Language Models. In *Proceedings of ICML*.

Gutierrez, B. J.; Shu, Y.; Gu, Y.; Yasunaga, M.; and Su, Y. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831*.

Izacard, G.; Caron, M.; Hosseini, L.; Riedel, S.; Bojanowski, P.; Joulin, A.; and Grave, E. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research*.

Jiang, H.; Wu, Q.; Lin, C.-Y.; Yang, Y.; and Qiu, L. 2023a. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of EMNLP*.

Jiang, H.; Wu, Q.; Luo, X.; Li, D.; Lin, C.-Y.; Yang, Y.; and Qiu, L. 2023b. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. *arXiv preprint arXiv:2310.06839*.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *Proceedings of ICLR*.

Karpukhin, V.; Oguz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; and Yih, W. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of EMNLP*.

Khattab, O.; and Zaharia, M. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of SIGIR*.

Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-t.; Rocktäschel, T.; Riedel, S.; and Kiela, D. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*.

Li, M.; Song, F.; Yu, B.; Yu, H.; Li, Z.; Huang, F.; and Li, Y. 2023a. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of EMNLP*.

Li, Y.; Dong, B.; Lin, C.; and Guerin, F. 2023b. Compressing Context to Enhance Inference Efficiency of Large Language Models. In *Proceedings of EMNLP*.

Liang, Y.; Wu, C.; Song, T.; Wu, W.; Xia, Y.; Liu, Y.; Ou, Y.; Lu, S.; Ji, L.; Mao, S.; Wang, Y.; Shou, L.; Gong, M.; and Duan, N. 2023. TaskMatrix.AI: Completing Tasks by Connecting Foundation Models with Millions of APIs. *arXiv preprint arXiv:2303.16434*.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; Zhang, S.; Deng, X.; Zeng, A.; Du, Z.; Zhang, C.; Shen, S.; Zhang, T.; Su, Y.; Sun, H.; Huang, M.; Dong, Y.; and Tang, J. 2023. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Welleck, S.; Majumder, B. P.; Gupta, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*.

Meng, X.; Wang, S.; and Fang, Y. 2026. SkillRAE: Agent Skill-Based Context Compilation for Retrieval-Augmented Execution. *arXiv preprint arXiv:2605.10114*.

Nogueira, R.; and Cho, K. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Patil, S.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2023. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. *arXiv preprint arXiv:2307.16789*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of EMNLP-IJCNLP*.

Robertson, S.; and Zaragoza, H. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*.

Sarthi, P.; Abdullah, S.; Tuli, A.; Khanna, S.; Goldie, A.; and Manning, C. D. 2024. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. In *Proceedings of ICLR*.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*.

Shen, Y.; Song, K.; Tan, X.; Li, D.; Lu, W.; and Zhuang, Y. 2023. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. In *Advances in Neural Information Processing Systems*.

Shi, W.; Min, S.; Yasunaga, M.; Seo, M.; James, R.; Lewis, M.; Zettlemoyer, L.; and Yih, W.-t. 2023. REPLUG: Retrieval-Augmented Black-Box Language Models. *arXiv preprint arXiv:2301.12652*.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

Wang, L.; Yang, N.; Huang, X.; Jiao, B.; Yang, L.; Jiang, D.; Majumder, R.; and Wei, F. 2022. Text Embeddings by Weakly-Supervised Contrastive Pre-training. *arXiv preprint arXiv:2212.03533*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*.

Xiao, S.; Liu, Z.; Zhang, P.; and Muennighoff, N. 2023. C-Pack: Packaged Resources to Advance General Chinese Embedding. *arXiv preprint arXiv:2309.07597*.

Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Hua, T.; Cheng, Z.; Shin, D.; Lei, F.; Liu, Y.; Xu, Y.; Zhou, S.; Savarese, S.; Xiong, C.; and Zhou, T. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems*.

Xiong, L.; Xiong, C.; Li, Y.; Tang, K.-F.; Liu, J.; Bennett, P. N.; Ahmed, J.; and Overwijk, A. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *Proceedings of ICLR*.

Xu, F.; Shi, W.; and Choi, E. 2023. RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation. *arXiv preprint arXiv:2310.04408*.

Yang, J.; Jimenez, C. E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; and Press, O. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of ICLR*.

Zheng, Y.; Zhang, Z.; Ma, C.; Yu, Y.; Zhu, J.; Wu, Y.; Xu, T.; Dong, B.; Zhu, H.; Huang, R.; and Yu, G. 2026. SkillRouter: Skill Routing for LLM Agents at Scale. *arXiv preprint arXiv:2603.22455*.

Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; and Chi, E. 2023a. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *Proceedings of ICLR*.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Bisk, Y.; Fried, D.; Alon, U.; and Neubig, G. 2023b. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854*.