# METASKILL: REUSABLE PROBLEM-SOLVING METHODS FOR ADAPTIVE SKILL ORCHESTRATION IN LLM AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Reusable skills provide specialized procedures, but large language model (LLM) agents must still determine how to organize and adapt their use. We introduce METASKILL, a two-layer framework for reusing problem-solving methods across tasks and domains. The method layer selects from 26 manually authored methods and binds abstract roles to task objects. The execution layer uses the instantiated method to identify intermediate objectives, coordinate native skills, and guide repair or method reselection. On 89 SkillsBench tasks and 995 ToolHop examples, MetaSkill outperforms both evaluated baselines in all six configurations, with accuracy gains of 3.51–13.48 percentage points over the strongest baseline. Ablations support explicit method-guided orchestration beyond direct method injection and further gains from method reselection.

## 1 INTRODUCTION

Reusable skills extend LLM agents' tool use (Yao et al., 2023b) with specialized procedures and complete workflows. Their effective use requires deciding which intermediate results to obtain, how to coordinate capabilities, and when to revise the plan. These decisions can share a problem-solving method across task types and domains, even when the concrete skills differ.

For example, economic detrending and video motion analysis both estimate an interfering component before interpreting residual variation. A trend and camera motion play analogous roles but require different estimation procedures. The reusable content is the principle and its role relationships: changing the interference estimate requires revisiting dependent conclusions.

Reasoning-template retrieval (Yang et al., 2024) and dependency-aware skill composition (Li et al., 2026a) provide foundations for this reuse. We introduce METASKILL, which connects methodological guidance to concrete skill use through two layers. The method layer selects an applicable method and binds its abstract roles to task objects. The execution layer uses method rules and current evidence to determine intermediate objectives, coordinate native skills, and revise execution or reselect the method from feedback.

Experiments on 89 SkillsBench tasks and 995 ToolHop examples show higher accuracy than BoT and AgentSkillOS in all six evaluated configurations. On SkillsBench, the full framework exceeds direct method injection by 10.11 percentage points in mean task score; disabling method reselection reduces the score by 4.50 points. ToolHop gains are concentrated on longer dependency chains.

## 2 RELATED WORK

### 2.1 AGENT SKILLS AND CAPABILITY REUSE

Tool-use research establishes how agents acquire and invoke external capabilities. Toolformer learns when and how to call APIs through self-supervised training (Schick et al., 2023), while ToolLLM combines tool-use instruction tuning with search over API-call sequences (Qin et al., 2024). Skill-based agents extend reuse to executable behaviors and workflows: Voyager accumulates retrievable behaviors (Wang et al., 2023), and Agent Workflow Memory induces routines from action trajec-



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.

tories (Wang et al., 2025). ExpeL extracts reusable insights from experience (Zhao et al., 2024), while AgentSkillOS organizes skill discovery and dependency-aware composition (Li et al., 2026a). These approaches supply capabilities and experience that can support future tasks. MetaSkill adds explicit methodological guidance for deciding which intermediate results those capabilities should produce and how their dependencies should be checked.

### 2.2 Methodological Guidance and Reusable Reasoning Structures

Chain-of-thought prompting exposes intermediate reasoning steps (Wei et al., 2022); Tree of Thoughts adds search and self-evaluation over alternative reasoning paths (Yao et al., 2023a). Reusable guidance further structures this process. SELF-DISCOVER adapts reasoning modules into task-specific structures (Zhou et al., 2024), Buffer of Thoughts retrieves and instantiates thought templates (Yang et al., 2024), and Chain of Methodologies selects human-defined methodologies as reasoning unfolds (Liu et al., 2025). Feedback also supports revision: Reflexion stores verbal reflections for subsequent attempts (Shinn et al., 2023), and AdaPlanner refines plans from execution feedback (Sun et al., 2023). MetaSkill connects methodological guidance to explicit task roles, skill-level objectives, and checks that govern local repair or method reselection.

### 2.3 Higher-Level Skills and Methodological Knowledge

SkillX distills trajectories into planning, functional, and atomic skills (Wang et al., 2026a); SkillPyramid consolidates hierarchical skills through composition and validation (Xiong et al., 2026). EffiSkill uses meta-skills to diagnose and compose code-optimization operators (Wang et al., 2026c), while Skill-MAS evolves principles for task decomposition and multi-agent workflow construction (Lin et al., 2026). At the system level, ADAS searches over code-defined agents (Hu et al., 2025), AFlow optimizes agentic workflows (Zhang et al., 2025b), and MaAS learns a supernet for query-dependent agent architectures (Zhang et al., 2025a). TROVE revises skill routes using runtime evidence while retaining valid continuations (Wang et al., 2026b). MetaSkill represents cross-task strategies through applicability conditions, role relationships, and decision rules. Binding these abstractions to task objects links reusable methodological knowledge to adaptive skill execution, even when tasks require different concrete procedures.

## 3 Problem Formulation

**Tasks and capabilities.** A task instance is

$$\tau = (x, \mathcal{S}_\tau, E, B), \tag{1}$$

where $x$ specifies instructions, inputs, constraints, and deliverables; $\mathcal{S}_\tau$ is the available native skill library; $E$ supplies tools and an interactive environment; and $B$ limits execution resources. The agent must interpret the requirements, obtain intermediate results, and produce a satisfying deliverable within this budget.

A native skill provides a capability description, procedural guidance, and optional resources. It may cover a complete workflow, including planning and validation, and can support multiple tasks. The agent interprets this guidance through reasoning and tools; a skill need not expose a single callable function.

**Reuse across tasks.** Specialized procedures change with the task, but their reasoning principles can recur. Vehicle rebalancing and transaction analysis involve different entities and operations, yet both require translating rules into explicit conditions and checking conclusions against them. Transferring this strategy requires identifying the relevant state and constraints in the new task, then selecting capabilities to construct and validate the representation.

Decision principles must transfer with these roles. For example, fixing every vehicle's initial load to zero narrows a rebalancing task that permits a bounded initial load. Further solver effort would still optimize the restricted model. A reusable method should therefore guide problem interpretation and checking, as well as the organization of operations.

**Objective.** We seek a shared method library $\mathcal{M}$ whose role relationships and decision principles can guide the use of $\mathcal{S}_\tau$ across tasks. Instantiation maps these abstractions to concrete entities and intermediate objectives. Execution observations then determine whether to continue, repair the current work, or reconsider its interpretation. Section 4 defines the representation and execution process; our experiments compare it with agents already equipped with native skills.

# 4 Design and Methodology

MetaSkill connects reusable problem-solving methods with available skills through two layers (Figure 1). The method layer selects and instantiates manually authored definitions; the execution layer uses them to organize concrete capabilities. Feedback supports local repair or method reconsideration. Library construction occurs offline, while instantiation and adaptation occur during task execution.

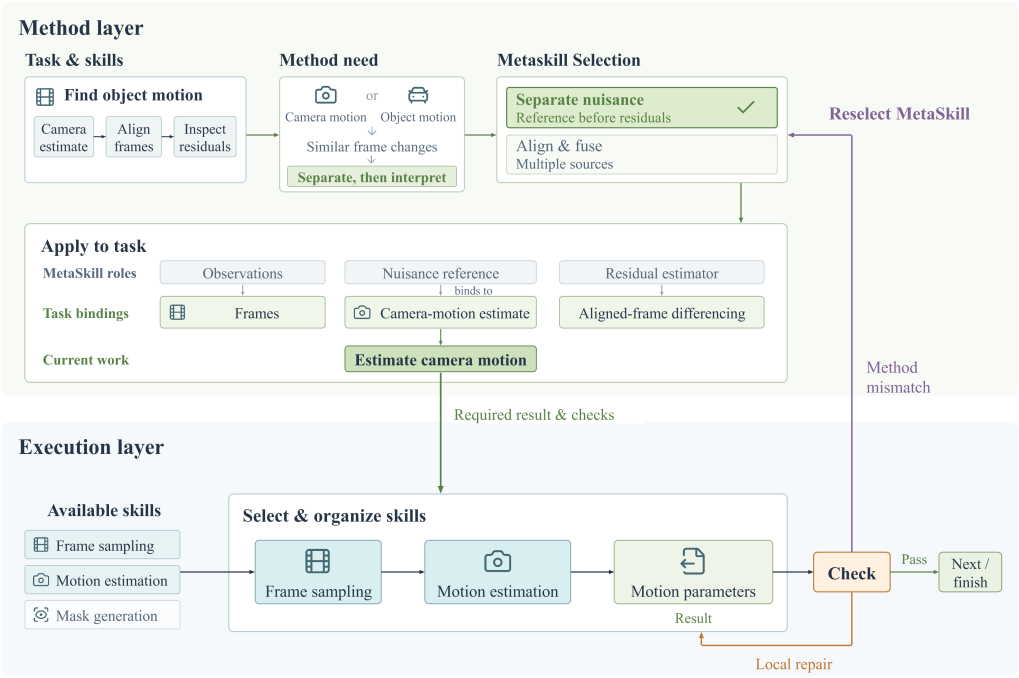



Figure 1: Method selection and execution for video motion analysis. Task and skill analysis yields requirements for comparing methods. Gray role slots bind to task objects and procedures (light green). Obtaining the camera-motion estimate defines the current work (dark green), realized through native skills. Feedback supports local repair or method reselection.

## 4.1 Method Layer: Abstraction and Task Instantiation

### 4.1.1 Manual Method Extraction and Representation

We derive methods from public task instructions and accompanying skill procedures. Instructions establish goals and constraints; procedures reveal operations, dependencies, checks, and recovery actions. Extraction identifies why these operations are necessary and which relationships transfer across tasks.

For example, economic detrending removes long-term trends before analyzing cyclical variation, while video analysis compensates for camera motion before identifying independently moving objects. Both instantiate *separate–nuisance–estimate*: establish an interfering component before interpreting residuals. The shared roles are observations, nuisance reference, target, and residual estimator. Filters, alignment algorithms, and thresholds remain with the native skills.

We encode these relationships as role slots, state variables, and policy rules. Source constraints and failure analysis provide applicability conditions, invariants, review checks, repair rules, and termination criteria. In the example, checks examine both removed and remaining variation; changing the nuisance estimate reopens dependent results. Table 1 illustrates the resulting definition.

Table 1: Extracted definition of *separate–nuisance–estimate*, abbreviated. ✓ marks fields included in its discovery card.

| Field | Example content | Card |
|---|---|---|
| Identifier | `separate-nuisance-estimate` | ✓ |
| Description | Account for nuisance effects before estimating the target from residuals. | ✓ |
| Entry conditions | Mixed target and nuisance effects; a defensible nuisance estimate. | ✓ |
| Exclusions | Do not remove variation merely because it is inconvenient. | ✓ |
| Role slots | Observations, target, nuisance reference, residual estimator. | |
| State variables | Estimate and version, valid region, residuals, uncertainty, dependent results. | |
| Policy rules | Establish the reference; inspect removed and remaining variation. | |
| Invariants | Preserve the target; unexplained residuals alone do not identify their cause. | |
| Review checks | Check sensitivity to the reference and possible removal of target variation. | |
| Repair rules | Revise the estimate or valid region; recompute affected results. | |
| Termination | Deliver supported estimates, or explain why separation remains unresolved. | |

The discovery card contains the identifier, description, entry conditions, and exclusions for candidate comparison. The full definition supplies application rules, stored in `method.json` and rendered as `SKILL.md`, alongside source and validation metadata. Appendix A describes the schema.

#### 4.1.2 Selecting and Instantiating a Method for a Task

**Identify the needed guidance.** The agent examines task goals, constraints, and relevant native procedures to identify dependencies, shared results, and revision points. It considers what these relationships mean for the goal: an estimate may remove interference in one task and support prediction in another. Method selection therefore depends on the purpose of the operations, as well as their ordering.

**Compare candidates.** The agent compares discovery cards, then reads relevant definitions to check applicability and useful decision guidance. In video analysis, *separate–nuisance–estimate* addresses the dependency between camera-motion estimation and residual interpretation; alignment and fusion would instead suit combining multiple observations. When native procedures already provide sufficient guidance, the agent can proceed without an added method.

**Bind roles to task objects.** The selected method is instantiated through a concrete role mapping. For video analysis, observations become frames, the nuisance reference becomes estimated camera motion, the target becomes independently moving objects, and the residual estimator computes compensated-frame differences. Revising the motion estimate consequently reopens dependent judgments. The agent records the method, selection rationale, role mapping, and scope: either the whole task or a subproblem.

### 4.2 Capability Execution: Realization and Feedback

#### 4.2.1 Selecting Skills and Organizing Their Execution

Execution identifies the next required result from task constraints, current evidence, and method rules. We call each such unit of work a method "node." Nodes arise as execution progresses and can use multiple skills, invoke a complete native workflow, or reuse existing results. Without an active method, task requirements and native procedures guide the same process.

**Select capabilities for the required result.** The agent checks candidate skills' capabilities, inputs, outputs, and conditions of use. Summaries support discovery; full definitions specify procedures and

restrictions. Establishing camera motion, for instance, can require a sampling-and-indexing skill followed by an egomotion-estimation skill. These skills are complementary, whereas two estimators may offer alternative implementations. The agent chooses among alternatives according to available inputs and task constraints.

**Coordinate dependencies and shared results.** The agent orders operations, specifies exchanged outputs, and maintains consistency across them. Sampling indices must correspond to the frames used for motion estimation, compensation, and mask generation. Missing inputs or incompatible outputs trigger revised skill selection or ordering. A skill can support several nodes, and a node can combine several skills; their relationship is determined by the required results.

**Check the outcome.** Before execution, the agent records the expected result, selection rationale, and conditions for changing the plan. It then checks the outcome against task requirements and method rules. Motion parameters, for example, must support valid compensation and residual analysis. Sufficient results enable progress; failures initiate the feedback process below.

#### 4.2.2 Feedback-Based Repair and Method Reselection

**Localize and repair.** The agent diagnoses the scope of a failed check. Missing inputs, incorrect arguments, and unsuitable implementations call for execution changes; misinterpreted objects require revised role bindings. In video analysis, an unreliable motion estimate can motivate better sampling or a different estimator. Valid intermediate results are retained, while affected dependencies are checked again or recomputed.

**Reconsider the method.** Method-level revision is triggered when entry conditions cannot be met, an exclusion applies, or the method's checks and repairs cannot support the required judgment. The agent records the conflicting assumption, attempted repairs, and missing evidence, then compares alternatives against the unresolved problem. If residuals remain compatible with both compensation error and true motion, *contrast–test–eliminate* may guide a discriminating test. Selecting it requires feasible observations with different predicted outcomes and an explanation of how they would change the next action. The agent can also return to native procedures when these address the gap.

**Resume with updated guidance.** A method change updates role bindings and checks before execution resumes. Existing frames and sampling indices may remain valid, while judgments based on unreliable compensation require reassessment. The agent records which results are retained or reopened. Execution ends when task and applicable method checks pass, or when the budget or feasible work is exhausted. Algorithm 1 summarizes this process.

**Algorithm 1** Task execution with an existing MetaSkill library

```
Require: Task, available skills, method library, environment, and budget
Examine requirements and skill dependencies; select a method or none.
Record the selection rationale and bind roles when a method is selected.
while budget permits and feasible work remains do
    Identify the next result using task requirements, current evidence, and method rules.
    Select skills by capabilities and prerequisites, or reuse valid results.
    Organize dependencies and shared outputs; revise skill choices as needed.
    Record the expected result and change conditions; execute and check the outcome.
    if final checks pass for task requirements and applicable method rules then
        return deliverables and execution records
    else if the outcome is insufficient then
        Diagnose the failure using feedback and earlier repair outcomes.
        if the evidence calls for reconsidering the method then
            Compare alternatives, including no added method, against the unresolved problem.
            Update the method, role bindings, and checks when justified and permitted.
        else
            Revise inputs, skills, execution, task interpretation, or role bindings as needed.
        end if
        Keep valid results; reopen affected conclusions and recompute as needed.
    end if
end while
return available results and unresolved conditions
```

# 5 EXPERIMENTS

## 5.1 EXPERIMENTAL SETUP

**Benchmarks.** Our SkillsBench evaluation combines public tasks from the benchmark's main collection (`tasks`) and supplementary collection (`tasks-extra`) (Li et al., 2026b). The source snapshot contains 94 main tasks and 5 supplementary tasks. After excluding 10 tasks with environment construction or setup problems, the evaluation set contains 89 tasks: 87 from the main collection and 2 from `tasks-extra`. We use this set for both the main comparison and the ablation study. We also evaluate on 995 ToolHop examples, which test multi-hop tool use (Ye et al., 2025).

**Baseline methods and configurations.** We compare MetaSkill with Buffer of Thoughts (BoT) (Yang et al., 2024) and AgentSkillOS (Li et al., 2026a). On SkillsBench, we run the comparison using Claude Code and PI Agent, each paired with DeepSeek-V4.1-Flash and GPT-4.1-mini. On ToolHop, we compare the three methods using GPT-4.1-mini and DeepSeek-V4.1-Flash. The ablation study uses PI Agent with DeepSeek-V4.1-Flash to examine native skills, direct method injection, and method reselection.

**Metrics.** The main SkillsBench comparison reports task accuracy, while ToolHop reports strict accuracy over its 995 examples. Both main-result tables also report average token consumption per task in thousands (k), where 1k equals 1,000 tokens. We compute this metric by dividing total token consumption by the number of evaluated tasks and by 1,000. The ablation table reports mean task scores, including partial credit. Accuracy and task scores are expressed as percentages, with higher values indicating better performance. Differences below are computed from the values displayed in the tables and reported in percentage points.

## 5.2 MAIN RESULTS

Tables 2 and 3 summarize the main comparison on SkillsBench and ToolHop, respectively.

**Overall performance.** MetaSkill achieves the highest accuracy in all six evaluated configurations. On SkillsBench with DeepSeek-V4.1-Flash, it reaches 68.54% on Claude Code and 65.17% on PI Agent, improving over the strongest baseline by 13.48 and 4.50 percentage points. With GPT-4.1-mini, the corresponding accuracies are 16.85% and 17.98%, with gains of 5.61 and 5.62 points. On

Table 2: Accuracy (%) and average token usage per task (k; 1k = 1,000 tokens) on the 89-task SkillsBench subset. Best accuracy within each configuration is shown in bold.

| | Claude Code | | PI Agent | |
|---|---|---|---|---|
| **Method** | **Acc. (%)** | **Tokens (k)** | **Acc. (%)** | **Tokens (k)** |
| *DeepSeek-V4.1-Flash* | | | | |
| BoT (Yang et al., 2024) | 50.56 | 2624.09 | 60.67 | 1323.48 |
| AgentSkillOS (Li et al., 2026a) | 55.06 | 3208.24 | 51.69 | 2327.29 |
| **MetaSkill** | **68.54** | 4255.59 | **65.17** | 2487.53 |
| *GPT-4.1-mini* | | | | |
| BoT (Yang et al., 2024) | 11.24 | 923.11 | 12.36 | 880.15 |
| AgentSkillOS (Li et al., 2026a) | 10.11 | 1020.01 | 8.99 | 830.75 |
| **MetaSkill** | **16.85** | 811.41 | **17.98** | 857.46 |

Table 3: Strict accuracy (%) and average token usage per example (k; 1k = 1,000 tokens) on 995 ToolHop examples. Best accuracy for each model is shown in bold.

| | GPT-4.1-mini | | DeepSeek-V4.1-Flash | |
|---|---|---|---|---|
| **Method** | **Acc. (%)** | **Tokens (k)** | **Acc. (%)** | **Tokens (k)** |
| BoT (Yang et al., 2024) | 47.24 | 18.87 | 61.21 | 33.27 |
| AgentSkillOS (Li et al., 2026a) | 41.61 | 37.45 | 57.19 | 77.18 |
| **MetaSkill** | **51.96** | 21.48 | **64.72** | 38.43 |

ToolHop, MetaSkill obtains 51.96% with GPT-4.1-mini and 64.72% with DeepSeek-V4.1-Flash, exceeding the strongest baseline, BoT, by 4.72 and 3.51 points. The advantage thus extends across agent frameworks, models, and both benchmarks.

**Token consumption.** The accuracy gains have different resource profiles across configurations. On SkillsBench with DeepSeek-V4.1-Flash, MetaSkill uses more tokens than either baseline. With GPT-4.1-mini on Claude Code, it achieves higher accuracy with the lowest average token consumption (811.41k). On ToolHop, MetaSkill uses fewer tokens than AgentSkillOS under both models, but more than BoT. We next examine the task types and error patterns underlying the accuracy differences.

### 5.3 Results Analysis

We examine where the improvements arise using the per-example ToolHop records underlying Table 3. We group examples by their annotated reference-chain length and domain, then examine answer types and failure outcomes. Detailed execution cases are presented separately in Appendix C.

**Dependency length and task type.** Table 4 shows that the largest advantage occurs on the 214 examples with five to seven reference steps. MetaSkill solves 150, compared with 122 for BoT and 109 for AgentSkillOS, reaching 70.09% accuracy. The 28 additional correct answers relative to BoT account for 80% of their net 35-answer difference over the full dataset. In contrast, MetaSkill solves two fewer three-step examples than BoT and nine more four-step examples. The largest gains therefore occur on longer sequences of dependent results.

The domain results further localize this pattern. MetaSkill obtains 57.14% on computing tasks, compared with 28.57% for BoT and 26.53% for AgentSkillOS, and 81.42% on film tasks, compared with 75.41% and 72.68%. However, BoT leads on history and AgentSkillOS leads on literature. Performance varies with the domain and its mix of dependency lengths and answer transformations.

Table 4: ToolHop strict accuracy (%) with DeepSeek-V4.1-Flash by reference-chain length and domain. Each panel partitions all 995 examples. Domains with fewer than 30 examples are grouped as Other.

| Group | N | BoT | AgentSkillOS | MetaSkill |
|---|---|---|---|---|
| *Reference steps* | | | | |
| 3 | 428 | **60.51** | 54.91 | 60.05 |
| 4 | 353 | 64.59 | 63.74 | **67.14** |
| 5–7 | 214 | 57.01 | 50.93 | **70.09** |
| *Domain* | | | | |
| Computing | 49 | 28.57 | 26.53 | **57.14** |
| Film | 183 | 75.41 | 72.68 | **81.42** |
| Genealogy | 138 | 34.78 | 27.54 | **36.96** |
| History | 146 | **44.52** | 41.78 | 41.78 |
| Linguistics | 37 | 62.16 | 45.95 | **70.27** |
| Literature | 45 | 75.56 | **84.44** | 82.22 |
| Mathematics | 56 | 71.43 | 71.43 | **73.21** |
| Other | 310 | 70.97 | 65.81 | **71.29** |
| Publishing | 31 | 87.10 | 80.65 | **96.77** |

Grouping by answer type yields a related observation. On 164 string-answer examples, MetaSkill achieves 60.37%, versus 44.51% for BoT and 40.85% for AgentSkillOS. Its advantage is smaller on 602 numeric-answer examples: 66.45%, versus 64.95% and 62.79%. On 165 date-answer examples, MetaSkill reaches 58.18%, below BoT's 61.82%. At the example level, MetaSkill solves 75 questions that BoT misses but misses 40 that BoT solves; the corresponding counts against AgentSkillOS are 147 and 72. These paired outcomes yield net gains of 35 and 75 correct answers, respectively.

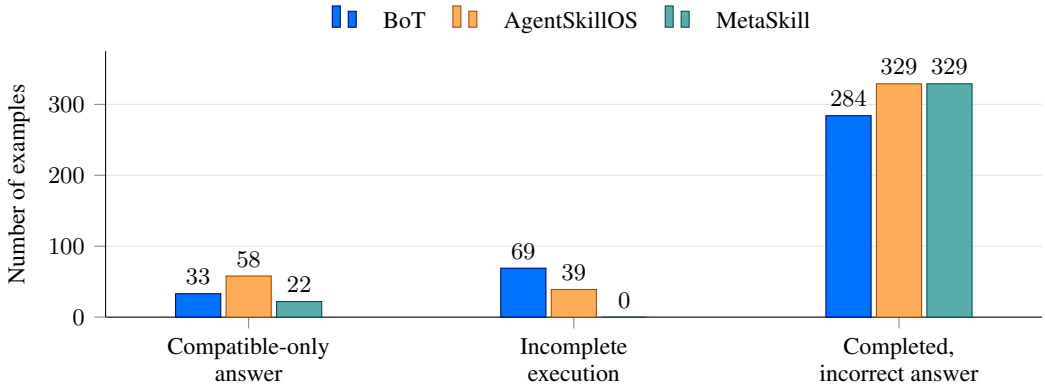



Figure 2: ToolHop error outcomes with DeepSeek-V4.1-Flash. Compatible-only answers pass the permissive scorer but fail strict scoring; the other two categories fail both. Total strict errors are 386 for BoT, 426 for AgentSkillOS, and 351 for MetaSkill.

**Failure outcomes.** Figure 2 separates strict errors into mutually exclusive outcome categories. A compatible-only result passes ToolHop's permissive evaluation but fails strict final-answer scoring; this can arise from answer representation or from the evaluator accepting tool feedback instead of the submitted answer. The remaining errors are separated by whether execution completed. MetaSkill has 22 compatible-only errors, compared with 33 for BoT and 58 for AgentSkillOS. Its remaining 329 errors occur despite completed execution. Most remaining errors thus concern the final answer after execution completes. Appendix C traces representative errors to specific operations.

**Mechanistic interpretation.** MetaSkill makes intermediate relationships and constraints explicit, directing checks toward the dependencies that determine the next action. The case studies in Ap-

pendix C show how this representation guides concrete interventions: checking component outputs to localize a date-parsing mismatch (Appendix C.1), correcting a counting parameter during renewed review (Appendix C.2), and preserving protected geometry while coordinating CAD extraction, checking, and repair (Appendix C.4). These examples connect method guidance to the choice of what to check and what to revise, while retaining valid intermediate results. The answer-formatting and semantic-review cases (Appendices C.3 and C.5) further identify final-answer representation and terminal computation as distinct review targets. Section 5.4 evaluates explicit orchestration and method reselection.

### 5.4 Ablation Study

We evaluate five configurations on the same 89-task SkillsBench set using PI Agent with DeepSeek-V4.1-Flash. Table 5 reports mean task scores, including partial credit, to examine native skills, explicit method-guided orchestration, and online method reselection.

Table 5: Ablation study on 89 SkillsBench tasks using PI Agent with DeepSeek-V4.1-Flash. Scores include partial credit. Drop denotes the decrease from the full system in percentage points (pp).

| Configuration | Mean task score (%) | Drop (pp) |
|---|---|---|
| w/o MetaSkill and native skills | 30.34 | 34.83 |
| w/o MetaSkill (skills only) | 52.81 | 12.36 |
| Skills + MetaSkills (direct injection) | 55.06 | 10.11 |
| w/o method reselection | 60.67 | 4.50 |
| **MetaSkill (full)** | **65.17** | — |

**Configurations.** The *skills-only* variant removes MetaSkill definitions and their method-level guidance; removing native skills as well gives the *w/o MetaSkill and native skills* condition. *Direct injection* supplies native skills and MetaSkill definitions as context and leaves planning and execution to the agent, rather than using the framework's explicit method-instantiation and orchestration procedure. The *w/o method reselection* variant retains initial method selection, task instantiation, and local execution repair, but disables changing the selected method during execution.

**Method knowledge and orchestration.** Native skills increase the score from 30.34% to 52.81%. Directly injecting MetaSkill definitions raises it to 55.06%, a further 2.25 percentage points. The full framework reaches 65.17%, exceeding direct injection by 10.11 points and skills alone by 12.36 points. This comparison supports explicit method-guided orchestration beyond providing method descriptions as additional context.

**Method reselection.** Disabling method reselection reduces the score from 65.17% to 60.67%, a drop of 4.50 percentage points. With local execution repair retained, this result supports adapting the selected method in addition to adjusting its concrete execution.

## 6 Conclusion

We introduced MetaSkill, a framework that connects reusable problem-solving methods to adaptive skill execution. By binding abstract method roles to task objects, MetaSkill guides intermediate objectives, skill orchestration, and feedback-based repair or method reselection. Experiments on SkillsBench and ToolHop show improvements over both baselines across all six evaluated configurations, with larger ToolHop gains on longer dependency chains. Ablations demonstrate the value of explicit orchestration beyond direct method injection and the contribution of online method reselection. These results support reusable methodological knowledge as a practical basis for coordinating agent capabilities. Future work will explore learning and refining method libraries from execution experience.

## AI USE STATEMENT

We used generative AI tools to assist with drafting and polishing the manuscript, including improving language, clarity, and organization, and preparing illustrative figures. The authors reviewed and revised the AI-assisted text and figures and take full responsibility for the final manuscript.

## REPRODUCIBILITY STATEMENT

Section 4 specifies the method representation and execution algorithm. Section 5.1 describes the benchmarks, model and agent configurations, and metrics. Appendices A and B detail the schema and runtime semantics, and Appendix C provides trace-based cases.

## REFERENCES

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. arXiv preprint arXiv:2603.02176, 2026a. URL https://arxiv.org/abs/2603.02176.

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

Hehai Lin, Qi Yang, and Chengwei Qin. Skill-MAS: Evolving meta-skill for automatic multi-agent systems. arXiv preprint arXiv:2606.18837, 2026. URL https://arxiv.org/abs/2606.18837.

Cong Liu, Jie Wu, Weigang Wu, Xu Chen, Liang Lin, and Wei-Shi Zheng. Chain of methodologies: Scaling test time computation without training. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5298–5312. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.276. URL https://aclanthology.org/2025.findings-acl.276/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

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*, volume 36, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/

075280-2537. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b5c8c1c117618267944b2617add0a766-Abstract.html.

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. 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. 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.

Zimu Wang, Yuling Shi, Mengfan Li, Zijun Liu, Jie M. Zhang, Chengcheng Wan, and Xiaodong Gu. EffiSkill: Agent skill based automated code efficiency optimization. arXiv preprint arXiv:2603.27850, 2026c. URL https://arxiv.org/abs/2603.27850.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Yuan Xiong, Ziqi Miao, Qian Chen, Lijun Li, Yequan Wang, Shizhu He, Jun Zhao, and Kang Liu. SkillPyramid: A hierarchical skill consolidation framework for self-evolving agents. arXiv preprint arXiv:2606.03692, 2026. URL https://arxiv.org/abs/2606.03692.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/cde328b7bf6358f5ebb91fe9c539745e-Abstract-Conference.html.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://arxiv.org/abs/2210.03629.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. ToolHop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2995–3021. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.150. URL https://aclanthology.org/2025.acl-long.150/.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852. PMLR, 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-Discover: Large language models self-compose reasoning structures. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4004. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e41efb03e20ca3c231940a3c6917ef6f-Abstract-Conference.html.

# A METHOD SCHEMA AND REPRESENTATIVE ENTRIES

Table 6 maps the semantic components in Section 4.1.1 to the detailed fields used in the manuscript's method schema. Identity, provenance, and validation metadata accompany the fields below. Role relationships and dependencies are expressed in natural-language role descriptions and policies.

Table 6: Core method components and their detailed schema fields.

| Core component | Schema fields | Function |
|---|---|---|
| Applicability | Entry conditions; exclusions | Specify when the method is relevant and which conclusions or uses are outside its scope. |
| Problem abstraction | Role slots; state variables | Identify entities and relationships to bind, and information relevant to the next decision. |
| Decision guidance | Policy rules; invariants; review checks; repair rules; termination | Guide the next required result, preserve reasoning constraints, and govern checking, revision, and completion. |

**Recover from redundancy.** The roles are unknowns, redundant observations, relationships, and tolerances. The method identifies independent relationships, determines which unknowns are identifiable, and solves them in dependency order. Remaining relationships provide cross-checks against source precision and units. Review examines uniqueness and conflicting constraints; a contradiction directs attention to the source relationship before a recovered value is revised. The method completes when identifiable required values have been recovered and checked, and remaining ambiguity or contradiction is described explicitly.

**Contrast–test–eliminate.** The roles are explanations, predictions, discriminating checks, and observations. The state retains live explanations, predicted outcomes, uncertainty, and unresolved alternatives. A useful check has outcomes that separate explanations under stated assumptions. An observation eliminates only explanations contradicted within the tested scope; insufficient sensitivity or compatible predictions leave alternatives unresolved. Review examines confounding, equivalent predictions, and dependencies of reused evidence.

**Parse–map–reconcile.** The roles are source fields, canonical entities or units, mapping rules, and a conflict policy. The state retains parsed records, mapping support, ambiguous fields, and a reconciliation ledger. Parsing syntax is separated from assigning meaning. Canonical values retain their source and transformation history, and an uncertain mapping remains uncertain rather than being silently filled. A changed mapping requires recomputation of affected downstream records.

**Rule–model–recompute.** The roles bind public rules to decision state, model predicates, and output checks. The method separates task constants from permitted decisions and temporary strategy restrictions. It then uses an appropriate modeling capability and recomputes the serialized result against the rules. Feasibility, incumbent quality, and proven optimality are reviewed separately. A feasible witness tests whether the formulation admits a particular allowed case; domain coverage requires checking the correspondence between the public rules and model constraints.

**Contract–decompose–integrate.** This method organizes component work around interfaces and dependencies. It calls for a minimal real end-to-end interaction before broad implementation and preserves a complete native procedure when suitable. Review exercises the final serialized artifact through its intended consumer, checking constructor, shape, and behavioral requirements in the actual integration path.

# B IMPLEMENTATION SEMANTICS

**Selection and state are model-interpreted.** The planning report selects one method identifier or none and supplies all required role bindings. The LLM interprets their suitability for the task,

while the runtime validates identifiers and nonempty bindings. Method state observations can use catalog state names or additional labels; the implementation records matched and unmatched labels separately for trace inspection.

**Method context during delegation.** An active method remains available in subsequent prompts during delegated work, and its review contract still applies. In delegate mode, the agent uses a native procedure to obtain the required result. Selecting no active method retains the full system's available catalog and execution guidance; a method-free experimental condition instead omits method resources from the agent's context.

**Evidence tracks applicability to artifacts.** The runtime records fingerprints of public file artifacts and dependencies used by checks. Supported directory deliverables are fingerprinted from their members and contents. A changed artifact or dependency invalidates affected evidence. Interpretation revision retains the obligation identity and source and reopens satisfaction for review under the revised reading. These records track evidence freshness and execution provenance.

**Repair limits are scoped.** The default permits one method switch and one targeted repair after a failed review or closure. Role rebinding within the same method is accounted for separately from switching methods. Routine tool debugging may occur within a coherent action. All work remains subject to the overall task budget. The final record retains available artifacts and internal closure status separately for external evaluation.

# C Case Studies

This appendix details the execution cases supporting the analysis in Section 5.3. ToolHop examples are identified by their dataset IDs and use the selected DeepSeek-V4.1-Flash trajectories underlying Table 3. The SkillsBench example is an archived PI Agent execution used to illustrate how a method coordinates multiple skills. We examine the task, observed failure or uncertainty, intervention, and resulting outcome.

## C.1 Diagnosing a Date-Parsing Mismatch

ToolHop example 202 asks for the digit sum of a retrieved death date. MetaSkill selects *trace-link-resolve* and records the chain from the named person to the father, the death date, and the numerical transformation. The first call to `date_digit_sum` on "11 July 1302" returns 8. Review identifies a conflict with the required computation, and repair requests component-level output: day sum 2, month sum 0, and year sum 6. This isolates the missing month contribution. Repeating the call with "11/07/1302" and explicit DMY formatting produces components 2, 7, and 6, yielding the correct answer 15. BoT submits no final answer and AgentSkillOS returns 18. The useful intervention is the choice of a diagnostic observation followed by a targeted input correction, while retaining the already established identity and date.

## C.2 Repairing a Letter-Counting Parameter

In example 151, both baselines return 0 repeated letters for the first name of a screenwriter. MetaSkill initially makes the same error. Its selected *trace-link-resolve* method prompts review of the film-credit join, which returns two screenwriters sharing the first name "David." Review reopens the chain because the identity is not uniquely resolved. During repair, the agent rechecks the credits and calls the letter-count tool with `ignore_case=true`; the tool returns two occurrences of "d," changing the answer from 0 to 1. Renewed checking reopens the tool call, and the repair stage corrects its case-sensitivity parameter.

## C.3 Checking the Final Answer Representation

Example 115 follows a five-step chain from film identification to a character's surname and a binary character code. All three methods reach the relevant code, but both baselines submit `01110000`, whereas the selected MetaSkill run submits `1110000`, following the experiment's instruction to

omit leading zeroes from numeric answers. MetaSkill receives strict credit and both baselines fail. This gain comes from final-answer representation: all three methods retrieve the same underlying code. End-to-end checking must cover both the evidence chain and the requested answer representation.

### C.4 Preserving Constraints in CAD Repair

The archived computer-aided design (CAD) task `ada-bathroom-plan-repair` illustrates a different use of the method layer. MetaSkill chooses *localize-preserve-repair*, mapping observed accessibility violations to editable regions and identifying the bathtub, stable element identifiers, and original CAD context as protected content. It coordinates drawing extraction, accessibility checks, and geometric repair. The saved change log records a 7.893-inch toilet displacement, grab-bar adjustments, and repositioning of the turning circle, while retaining the door and bathtub. The run produces both the repaired DXF and synchronized layout and change records, and receives an official score of 1.0; the archived PI Agent AgentSkillOS execution receives 0.0. The trace illustrates how explicit preservation constraints guide several skills toward a consistent artifact.

### C.5 A Failure of Semantic Validation

In ToolHop example 35, MetaSkill correctly resolves the film, producer, and surname, then accepts a combination-counting tool's output of 6. Its review passes because the calls form a supported chain, but the submitted answer still fails the benchmark. Here, repeating the successful retrieval path does not test whether the final operation implements the intended counting semantics. The case exposes a remaining error mode: validating evidence provenance without independently validating the meaning of the terminal computation.