Lost in Implicit Routing: How LLM Agents Fail to Load Relevant Skills at Scale

Anonymous ACL submission

Abstract

Reusable skills let LLM agents load task-specific instructions on demand, but they also make task execution depend on implicit routing over skill metadata. We test whether *implicit routing* scales by holding SkillsBench tasks and golden skills fixed while adding irrelevant skills. We find *skill routing collapse*: as irrelevant skills grow, agents increasingly fail to load the correct skills even though those skills remain available. Loaded-skill recall drops from 73.6% with no irrelevant skills to 27.4% with 1,000 irrelevant skills, while explicit skill selection still reaches 73.9% recall at the same scale. We then compare several explicit preselection strategies and derive practical insights for different skill-library sizes. Finally, we show why agents are lost in implicit routing. Masking irrelevant metadata largely restores golden-skill loading, and a plan-first prompt remains weaker than isolated preselection, indicating that the bottleneck comes from semantic interference among skill descriptions and instruction interference between routing and execution, not from context growth alone. These results can inform future work on skill protocols and evaluation for agents with growing skill libraries.

1 Introduction

Large language model (LLM) agents are no longer only dialogue systems. They now operate tools and files to complete multi-step workflows across domains such as software engineering, data analysis, document processing, and personal assistance (Yao et al., 2023; Liu et al., 2024b; Wang et al., 2024). As these agents become more capable, they also need reusable procedures: the same plotting routine, code-audit workflow, spreadsheet operation, or domain-specific calculation should not be rediscovered from scratch in every new task.

Agent skills are one increasingly common answer. A skill packages domain-specific procedures and supporting resources into a reusable module. Anthropic's Agent Skills formalize this idea as composable capability packages that equip agents with specialized instructions and resources (Anthropic, 2025; Zhang et al., 2025). The ecosystem is also growing quickly: a recent analysis of public Skills reports growth from 2,179 listed skills on January 16, 2026 to 40,285 on February 5, a net increase of 38,106 skills in 20 days (Ling et al., 2026). Skill selection is therefore becoming a scaling problem.

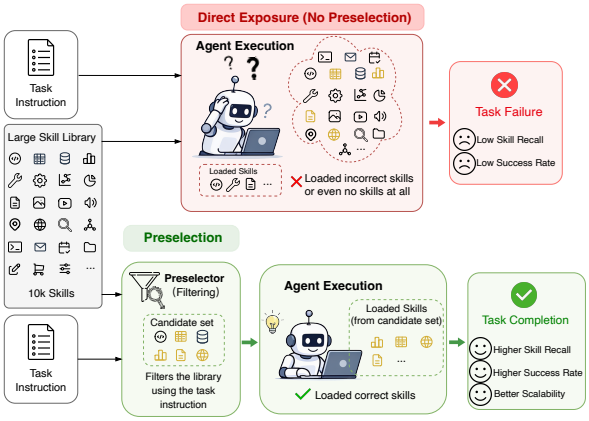

Figure 1: Lost in implicit routing. The relevant skills remain installed, but as irrelevant skills grow, agents increasingly fail to load them during implicit execution. Explicit routing recovers much of the missing skill recall.

In this paper, we use *skills* to refer specifically to Anthropic's Agent Skills and their progressive-disclosure design: the model first sees only names and descriptions, and opens the longer skill instruction file only if it decides the skill is relevant (Anthropic, 2026). In this design, routing is not a separate stage before task solving. The agent chooses whether to load skills and which skills to load while it is also planning actions, inspecting files, and completing the task. We call this execution-time selection process *implicit routing*. The design is sensible because full skill files can be long, but it also hides a strong assumption: an

agent can identify and invoke the few skills that matter from many skill descriptions while simultaneously solving the task.

We test this assumption with a controlled scaling experiment on SkillsBench (Li et al., 2026), a public benchmark that provides fixed task instructions, workspaces, verifiers, and relevant skills. For each task, we keep these components fixed and vary only the number of added irrelevant skills.

The result is a failure mode we call *skill routing collapse*: the right skills are installed, but the agent does not load them. Under implicit routing, loaded-skill recall drops from 73.6% with no irrelevant skills to 27.4% with 1,000 irrelevant skills, and the rising zero-load rate means agents increasingly tend not to load any skill at all. However, when asked to choose skills before execution, the same models retain 73.9% selected-skill recall with 1,000 irrelevant skills. This contrast is our main observation: relevant skills are often recognizable, but they are lost in implicit routing when skill loading is left as a subtask inside the execution loop.

End-to-end Pass@3 drops less sharply, from 51.6% to 38.6%, because strong agents can sometimes solve around missing skills. Therefore final task success alone is insufficient: it can make a broken skill system look healthier than it is. We make three contributions:

- We identify and measure *skill routing collapse*: agents become lost in implicit routing as the number of irrelevant skills grows, even when the correct skills remain available.
- We conduct a large-scale experimental comparison of preselection strategies and derive practical insights for different skill-library sizes.
- We show that being lost in implicit routing is driven by semantic interference from irrelevant skills and instruction interference from parallel routing and execution, rather than context growth alone.

2 A Controlled Test for Skill Routing

2.1 Task setup

We represent each task as $\tau = (I, F, V, G)$, where I is the task instruction, F is the task workspace, V is an executable verifier, and G is the set of relevant SkillsBench skills for the task. A run succeeds only if executing V after the agent finishes returns pass; no LLM judge is used. For a library-size setting s, the installed skill inventory is $S_s = G \cup D_s$, where D_s contains s irrelevant skills sampled independently of the task.

2.2 Experimental method

Figure 2 summarizes our experimental method. For each task, the golden skills are always present; we vary only the number of added irrelevant skills. This tests whether an agent can still find and load the needed skills as the metadata list grows.

In implicit routing, the execution agent sees the full metadata list and decides during execution whether to open any skill. Explicit routing adds a preselection step: a preselector receives the task and skill metadata, chooses a compact subset, and exposes only that subset to the execution agent. The progressive-disclosure protocol is otherwise unchanged: during execution, the agent still sees metadata first and must open a skill before using its full instructions. We also include a no-skill baseline.

We evaluate three preselection strategies. *LLM preselect* asks an LLM to directly choose skills from the full metadata list. *RAG preselection* retrieves skills by embedding similarity. *Hybrid preselection* uses retrieval to form candidates and an LLM selector to choose the final skills. Appendix I gives the execution and preselection pseudocode.

2.3 Metrics

We use the following main metrics. *Pass@3* is final task success, computed with the standard unbiased estimator over repeated runs. *Loaded-skill recall* is $|G \cap L|/|G|$, where L is the set of skills actually opened during execution. *Zero-load rate* is the percentage of execution trials with $|L| = 0$; it counts runs that load no skill at all. For explicit-routing probes, we report *selected-skill recall*, $|G \cap P|/|G|$, where P is the skill set selected before execution. Appendix F reports the corresponding precision metrics.

3 Main Findings

3.1 Experimental setup

SkillsBench tasks. We evaluate on SkillsBench (Li et al., 2026), an existing public benchmark for evaluating agent skills across diverse tasks. It provides the components needed for our controlled

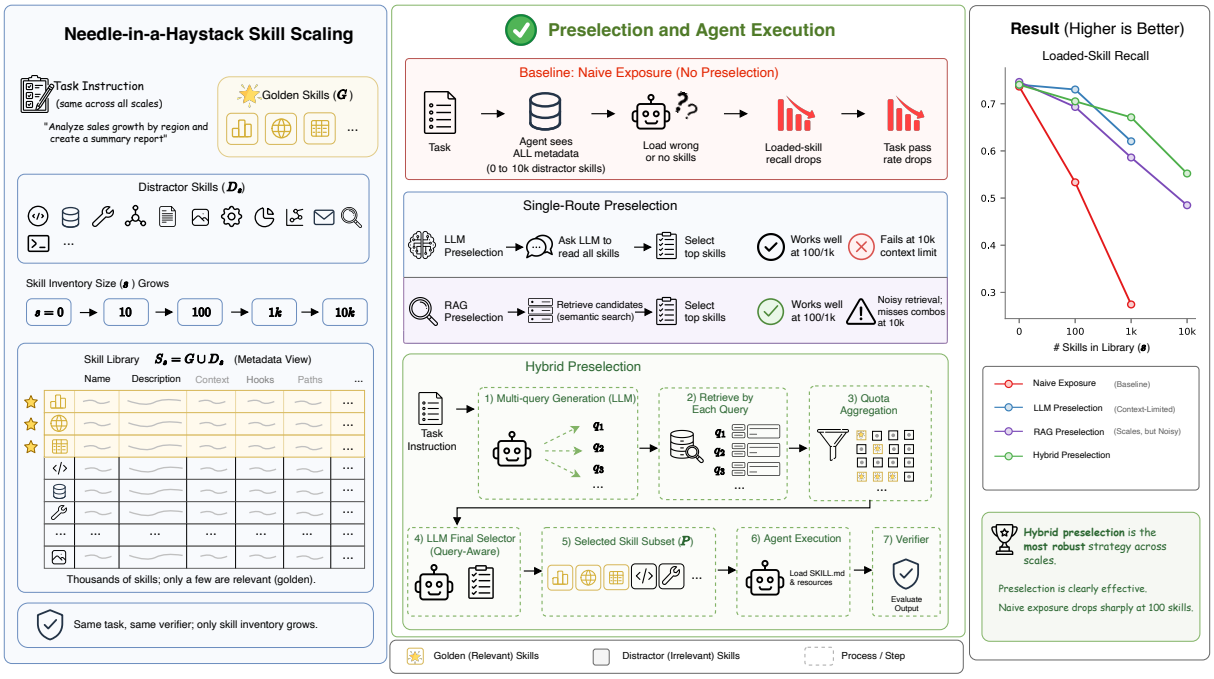

Figure 2: Overview of the controlled skill-scaling setup and routing strategies. Starting from an existing SkillsBench task, we keep the instruction, files, verifier, and relevant skills fixed while varying only the number of added irrelevant skills.

study: task instructions, workspace files, executable verifiers, and task-specific golden skills. Because five tasks have infrastructure or environment problems, we use the remaining 82 tasks in the main controlled study; the excluded tasks are listed in Appendix B. Runs are expensive: under implicit routing, one run averages 15.3m and about 485k input tokens with no added irrelevant skills, increasing to 17.0m and about 1.95M input tokens with 1,000 added irrelevant skills; output is about 13k tokens in both settings. Appendix A gives additional dataset information.

Skill-library construction. Each task has its own set of golden skills. Unless otherwise stated, these golden skills are always included in the installed skill library. We downloaded more than 20,000 skills from a skill market in leaderboard order, and randomly sample added irrelevant skills from this pool. All experiments use the same random seed, 42, so the sampled irrelevant skills are held fixed across models and routing strategies. We use s to denote the number of added irrelevant skills and evaluate $s \in \{0, 10, 100, 1000, 10000\}$. Thus every scale keeps the task, files, verifier, golden skills, and irrelevant-skill sample fixed; only the number of added irrelevant skills changes. All explicit preselection strategies expose at most five skills to the execution agent: LLM-based selectors may choose any number up to five, while RAG directly returns the top five by embedding similarity.

Models. Because each run is token-expensive, the main experiments use three open-weight execution models: GLM-5 (Zeng et al., 2026), Kimi K2.5 (Bai et al., 2026), and MiniMax M2.5 (MiniMax AI, 2026). We use Qwen text-embedding-v4 for RAG and hybrid-preselection embeddings. Appendix E reports a budget-limited validation on Claude Opus 4.6, Gemini 3.1 Pro, and GPT-5.4; its conclusions are consistent with the open-weight results.

Experimental scale. For each task and skill-library condition, we run each model three times and average over the task–model–run grid. The experiments in this paper collect more than 15,000 execution trajectories and use more than 10 billion model tokens.

3.2 Lost in Implicit Routing

3.2.1 Implicit loading degrades before task success

Figure 3 shows the implicit-routing results for the three open-weight models. In this setting, the execution agent sees the full metadata list and must decide during execution whether to open any skill. **Implicit routing collapses as irrelevant skills grow.** The evidence appears first in loaded-skill

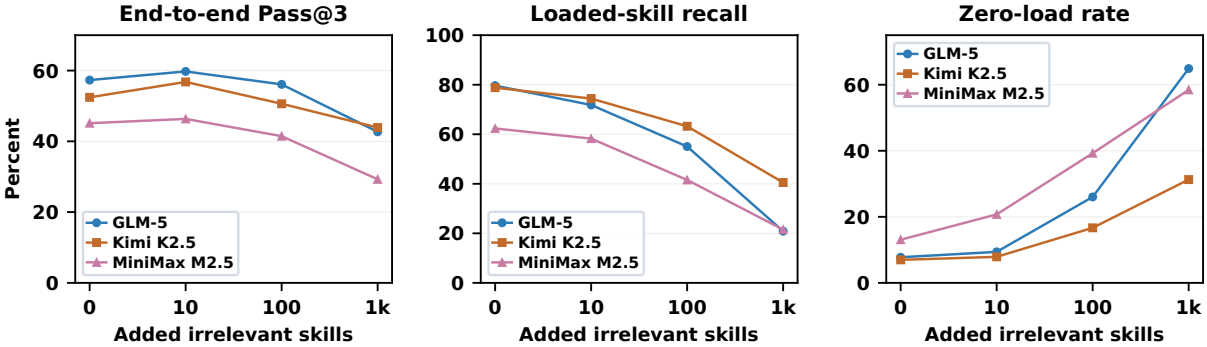

Figure 3: Implicit-routing scaling by execution model. (a) End-to-end Pass@3. (b) Loaded-skill recall. (c) Zero-load rate. Only the number of added irrelevant skills changes.

recall: with only 100 irrelevant skills, recall falls from 79.6% to 55.0% for GLM-5, from 78.8% to 63.2% for Kimi K2.5, and from 62.3% to 41.6% for MiniMax M2.5. Zero-load rises at the same time, so agents are not merely choosing wrong skills; they increasingly stop using the skill mechanism at all. The complete plotted values are listed in Appendix D.1.

Final task success moves less uniformly than skill loading. GLM-5, for example, keeps Pass@3 almost unchanged from 57.3% to 56.1% at 100 irrelevant skills even though its loaded-skill recall drops by 24.6 points. Kimi K2.5 also keeps Pass@3 close to its no-irrelevant-skill value at 100 irrelevant skills, while its loaded recall has already fallen by 15.6 points. These gaps show why final success alone can hide routing failure: stronger execution models can sometimes solve around missing skills, but the skill protocol is already failing. A budget-limited validation on three proprietary frontier models shows the same qualitative pressure; their Pass@3 often declines more slowly, but their loaded-skill recall still falls as irrelevant skills grow (Appendix E).

3.2.2 The same skills are recognizable when selected explicitly

A second comparison asks whether the relevant skills are intrinsically hard to recognize. Table 1 compares two measurements for the same models and scales: implicit loaded-skill recall during execution, and explicit LLM selected-skill recall before execution. **The same LLMs can still identify the needed skills when selection is explicit.** The gap is large. At 1,000 irrelevant skills, GLM-5 loads only 20.9% of golden skills under implicit routing, but explicitly selecting before execution reaches 82.5%. Kimi K2.5 shows the same pattern, 40.5% loaded versus 80.7% selected. Even for MiniMax M2.5, explicit selection remains much higher than implicit loading.

Model	Measure	0	10	100	1k
GLM-5	Implicit loaded	79.6	71.8	55.0	20.9
GLM-5	Explicit selected	98.4	93.1	91.7	82.5
Kimi K2.5	Implicit loaded	78.8	74.4	63.2	40.5
Kimi K2.5	Explicit selected	98.5	93.3	90.7	80.7
MiniMax M2.5	Implicit loaded	62.3	58.2	41.6	21.5
MiniMax M2.5	Explicit selected	98.3	88.0	83.6	58.4

Table 1: Implicit loaded-skill recall (%) versus explicit LLM selected-skill recall (%) by model and number of added irrelevant skills.

This comparison locates the bottleneck more precisely. The relevant skill metadata is often sufficient for an LLM to name the right skills when selection is made an explicit step. The failure appears when that selection is folded into the execution loop, where the agent is simultaneously reading files, planning commands, and deciding whether to invoke skills. This shows that agents often have the ability to choose the right skills, but become lost in implicit routing when skill selection competes with the many parallel demands of execution. The next section therefore analyzes explicit preselection as a controlled contrast: what changes when the visible skill set is narrowed before execution begins?

3.3 Analyzing Explicit Routing

We next analyze whether narrowing the visible skill set before execution can reduce skill routing collapse. The previous section showed that, when selection is requested explicitly, an LLM can often identify the correct skills before execution. This motivates a controlled comparison: first select a task-relevant skill subset, then expose only that subset to the agent during execution. We do not propose a new routing algorithm; instead, we com-

Model	w/o	0	10	100	1k	10k
Naive						
GLM-5	26.8	57.3	59.8	56.1	42.7	–
Kimi K2.5	34.1	52.4	56.8	50.6	43.9	–
MiniMax M2.5	26.8	45.1	46.3	41.5	29.3	–
LLM preselect						
GLM-5	26.8	65.9	57.3	57.3	58.5	–
Kimi K2.5	34.1	54.9	56.1	58.5	53.7	–
MiniMax M2.5	26.8	43.9	43.9	48.8	42.7	–
RAG						
GLM-5	26.8	57.3	48.8	53.7	54.3	47.6
Kimi K2.5	34.1	50.0	50.0	52.4	48.8	41.5
MiniMax M2.5	26.8	50.0	45.1	42.7	36.6	35.4
Hybrid preselection						
GLM-5	26.8	59.8	61.0	54.9	57.3	56.1
Kimi K2.5	34.1	48.8	56.1	52.4	52.4	56.1
MiniMax M2.5	26.8	50.0	43.9	42.7	48.8	39.0

Table 2: Per-model Pass@3 (%) by routing strategy and skill-pool size. The w/o column disables skill exposure entirely. Numeric columns give the number of added irrelevant skills on top of the task's golden skills. "–" indicates omitted infeasible settings.

pare several simple preselection strategies to understand when this design choice helps and what tradeoffs it creates for practical deployments.

3.3.1 Explicit routing across library sizes

Figure 4 compares implicit routing with LLM preselection, RAG preselection, and hybrid preselection. **Narrowing the visible skill set reduces the scale effect.** Explicit routing generally makes Pass@3, loaded-skill recall, and zero-load rate less sensitive to added irrelevant skills. The best strategy changes with scale: LLM preselection is strongest around 100 added skills, while hybrid preselection is strongest at 1,000 and remains usable at 10,000, where full-list LLM preselection is omitted. Appendix D.2 gives the aggregate values, and Table 2 gives the per-model Pass@3 breakdown.

Table 2 shows the same pattern across execution models: implicit routing degrades as the library grows, while exposing a selected subset often recovers part of the loss.

Table 3 compares preselection recall, which measures whether the preselector names the golden skills before execution. This differs from loaded-skill recall in Figure 4, which measures whether the execution agent later opens those skills. Recall should be read together with selected precision in Appendix F, Table 23. RAG has the highest recall at 10 and 100 added skills, partly because it always returns five skills; its selected precision is much lower at the same scales. Hybrid

Method	0	10	100	1k	10k
LLM	98.4	91.5	88.7	73.9	–
RAG	98.6	95.9	90.0	74.1	62.0
Hybrid	98.2	91.4	87.9	81.9	72.2

Table 3: Preselected-skill recall (%) by method and number of added irrelevant skills.

preselection is preferable at large scale: at 1,000 it exceeds both LLM preselection and RAG preselection in recall, and at 10,000 it improves over RAG from 62.0 to 72.2 while also selecting fewer irrelevant skills, as reflected by its higher selected precision in Table 23.

3.3.2 Token cost of routing

Table 4 separates LLM-side input and output tokens for both execution and preselection. We exclude embedding cost throughout because skill metadata embeddings are cached and the total embedding-model spend for this paper is below $1. The main tradeoff is that preselection adds a routing step but can sharply reduce the execution context, especially when the skill library is large. At 1,000 irrelevant skills, implicit routing reaches 1952k agent-input tokens, whereas LLM preselection adds a 36k-input routing call but reduces agent input to 497k, for 547k total tokens. Cost alone favors RAG: at 10,000 irrelevant skills it is the cheapest option at 469k total tokens because it uses no LLM selector call. However, Figure 4 shows that RAG also has the weakest large-scale performance. Hybrid preselection spends modestly more—547k total tokens at 10,000 irrelevant skills, including 6k preselection input and 2k output—but generalizes better at large scale.

3.3.3 Why hybrid preselection helps at 10k

A single RAG query based only on the task instruction can over-focus on one part of a multi-step task and miss other required capabilities. Hybrid preselection reduces this risk by asking an LLM to generate several task-focused search queries, retrieving from multiple semantic angles, and then applying a final LLM selector to the candidate pool. Table 5 compares the hybrid candidate pool with single-query RAG@100 before final selection. At 10,000 irrelevant skills, the hybrid candidate pool raises individual candidate recall from 88.2 to 91.8 and all-gold coverage from 78.0 to 86.2, reducing the chance that a needed skill is absent from the candidate pool. Additional details on candidate aggregation appear in Appendix H.

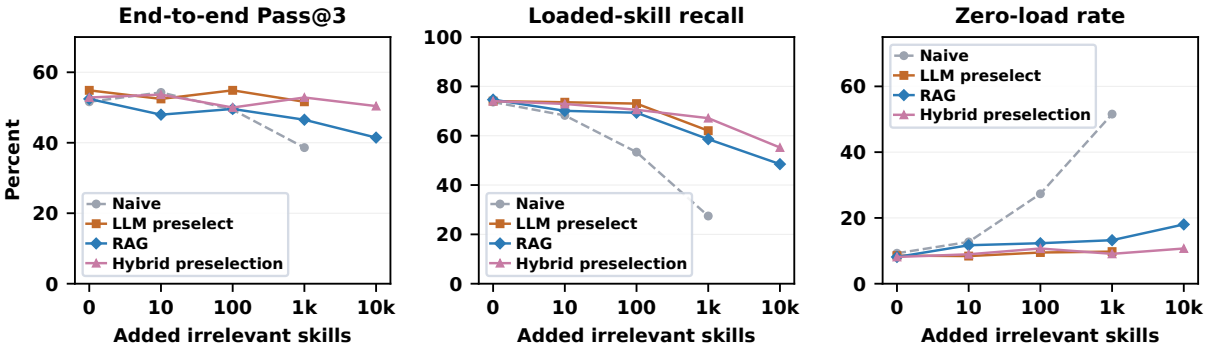

Figure 4: Skill scaling curves by routing strategy. (a) End-to-end Pass@3. (b) Loaded-skill recall. (c) Zero-load rate.

s	Strat.	A-In	A-Out	P-In	P-Out	Total
0	Naive	485k	13k	0k	0k	498k
0	LLM	481k	13k	0k	0k	494k
0	RAG	494k	13k	0k	0k	507k
0	Hybrid	499k	13k	0k	0k	513k
10	Naive	481k	13k	0k	0k	494k
10	LLM	495k	13k	1k	0k	509k
10	RAG	481k	13k	0k	0k	494k
10	Hybrid	502k	14k	1k	1k	519k
100	Naive	619k	13k	0k	0k	632k
100	LLM	490k	13k	4k	1k	508k
100	RAG	493k	13k	0k	0k	506k
100	Hybrid	476k	13k	6k	1k	497k
1000	Naive	1952k	13k	0k	0k	1966k
1000	LLM	497k	13k	36k	1k	547k
1000	RAG	490k	13k	0k	0k	503k
1000	Hybrid	533k	14k	6k	2k	554k
10000	RAG	456k	13k	0k	0k	469k
10000	Hybrid	525k	14k	6k	2k	547k

Table 4: Token decomposition by skill-pool size and strategy. A/P denote execution-agent/preselection tokens; In/Out denote input/output tokens.

s	SQ Rec.	Hybrid Rec.	Δ Rec.
0	99.4	99.4	+0.0
10	99.4	99.4	+0.0
100	99.4	99.4	+0.0
1000	93.6	96.3	+2.7
10000	88.2	91.8	+3.6
s	SQ All	Hybrid All	Δ All
0	98.8	98.9	+0.1
10	98.8	98.9	+0.1
100	98.8	98.9	+0.1
1000	87.8	92.0	+4.1
10000	78.0	86.2	+8.2

Table 5: Single-query RAG@100 versus the hybrid candidate pool. Values are percentages. Top: individual recall; bottom: all-gold coverage.

4 Why Agents Are Lost in Implicit Routing

4.1 Experimental setup

We analyze the mechanisms behind routing collapse with two controlled interventions on a 10-task subset at 1,000 irrelevant skills; the subset is used for budget reasons and listed in Appendix C. Each task is run three times with each of the three open-weight models, and all results in this section average over those model–run repetitions. **Masked** preserves the length and slot structure of the skill list while replacing each irrelevant skill's name, description, and location with length-matched `<mask>` placeholders. **Plan-first prompt** preserves the original metadata but adds an instruction to consider required skills before task execution. The corresponding prompt sketches appear in Appendix J.5. Table 6 reports these interventions alongside a no-extra-skills reference and explicit preselection on the same tasks.

4.2 Results and discussion

Skill metadata creates semantic interference. Table 6 shows that the masked condition keeps the context long, yet largely restores the no-extra-skills behavior and remains competitive with the strongest preselection rows. The intervention preserves the same skill slots and uses length-matched placeholders rather than simply deleting irrelevant metadata, so the improvement is not an artifact of a visibly shorter context. Thus the long metadata context by itself is not what causes routing collapse. The failure appears when added irrelevant skills carry meaningful descriptions that look like plausible alternative actions. When those descriptions are replaced by `<mask>` placeholders, the same long list produces much less interference with golden-skill loading.

Implicit routing suffers from instruction interference. The Plan-first prompt improves over the original condition, but remains weaker than explicit preselection. To isolate the initial routing decision, we extract planned skills from the agent's first response and compare them with isolated LLM preselection in Table 7. Plan-first recall is

Setting	Load Rec.	Zero	All-gold
No extra skills	73.5	5.6	52.2
Implicit routing, 1,000 irrelevant skills			
Original	18.6	62.2	6.7
Masked	63.9	23.6	49.4
Plan-first prompt	44.6	15.7	25.8
Explicit preselection, 1,000 irrelevant skills			
LLM	50.4	10.0	30.0
RAG	56.4	21.6	29.5
Hybrid	61.1	11.1	38.9

Table 6: Routing results (%) on the same 10-task subset. The no-extra-skills row uses only golden skills; all other rows use 1,000 irrelevant skills. Load Rec. is loaded-skill recall; Zero is the no-skill-load rate; All-gold is the percentage of runs in which all golden skills are loaded or selected.

Setting	Recall	All-gold
Plan-first prompt	39.9	20.5
LLM preselection	70.6	54.4

Table 7: Golden-skill selection quality (%) in the Plan-first response versus isolated LLM preselection on the same 10-task subset. Recall is computed against the task's golden skills.

lower than final loaded-skill recall because agents sometimes load useful skills later even when those skills are absent from the initial plan. The visible context for Plan-first and LLM preselection is nearly identical—the task instruction plus the full skill metadata. The main difference is the system prompt: LLM preselection asks only for skill selection, whereas Plan-first asks the agent to solve the task and explicitly requires it to choose skills before doing so. With the single skill-selection instruction, the LLM selects golden skills with much higher recall. In Plan-first, despite the explicit reminder to choose skills before task execution, the agent still misses many more golden skills. This recall drop indicates interference from parallel instructions: routing to skills and beginning task execution compete within the same response.

4.3 Case study

Plan-first execution can still become inconsistent, further supporting the instruction-interference explanation. Figure 5 shows two trace-level examples: the agent may name relevant skills but fail to load them, or it may proceed without planning skills at all.

These traces show that the plan-first instruction does not reliably isolate routing from task execution. Prompt nudges can mitigate this pressure, but reducing the visible skill set more directly addresses the semantic and instruction interference documented above.

Case 1: planned but not loaded

Task/model: *adaptive-cruise-control*, GLM-5.
Agent plan excerpt: "Let me load the relevant skills first: 1. ***pid-controller*** – For implementing PID control loops; 2. ***vehicle-dynamics*** – For vehicle motion simulation and cruise control modes; 3. ***simulation-metrics*** – For calculating performance metrics."
Actually loaded: *pid-controller*.
Missed: *vehicle-dynamics*, *simulation-metrics*, ... The run fails.

Case 2: no skill plan

Task/model: *data-to-d3*, MiniMax M2.5.
Agent plan excerpt: "Explore the input data files; set up the output directory; copy the input data; download D3.js v6; create the visualization with bubble chart and data table; add interactive features."
Actually loaded: none.
Missed: *d3-visualization*. The run fails.

Figure 5: Two trace cases under the plan-first prompt. Actually loaded merges explicit skill-loading events and direct skill-document reads.

5 Practical Recommendations

The preceding analysis suggests that reducing the visible skill set is more reliable than relying only on stronger prompt instructions. Because irrelevant skill descriptions interfere with both routing and task execution, practical systems should narrow the visible skill set before the execution loop whenever the library is no longer small.

The results suggest scale-dependent recommendations for skill preselection. For small curated libraries, implicit routing can remain usable, but systems should still log which skills are selected and loaded rather than relying only on final task success. Once the library approaches hundreds of skills, explicit preselection should become the default. Direct LLM preselection is a simple and strong choice when the full metadata list fits in context, because it keeps the agent focused on a compact skill subset before execution begins.

For libraries with thousands of skills or more, full-list LLM preselection becomes expensive or infeasible. In this regime, the hybrid preselection pattern—retrieval to construct a compact candidate set, followed by an LLM selector for final skill choice—is the strongest large-scale option in our experiments: it improves relevant-skill coverage while keeping the execution context small. In practice, skill systems should therefore switch from implicit routing to LLM preselection around the hundred-skill scale, and to hybrid preselection once the library reaches thousands of skills. Prompt reminders to think about skills can be useful as a lightweight safeguard, but they should complement rather than replace preselection.

6 Related Work

LLM agents and tool choice. LLM agents extend language models from single-turn response generation to multi-step interaction with tools and environments. ReAct couples reasoning with actions (Yao et al., 2023), Toolformer studies tool-call learning (Schick et al., 2023), and MRKL frames LLM systems as modular architectures combining language models with external knowledge and reasoning modules (Karpas et al., 2022). As the action space grows, later work studies large API collections and agent evaluation: Gorilla and ToolLLM focus on API selection and invocation over massive API sets (Patil et al., 2024; Qin et al., 2024), MetaTool evaluates whether models know when and which tools to use (Huang et al., 2024), and RAG-MCP studies retrieval-based tool discovery under prompt bloat (Gan and Sun, 2025). Recent work also shows that overlapping names and descriptions create tool-selection ambiguity and motivates tool merging or filtering (Liu et al., 2025a).

Skills versus tools. Skills are more general than callable tools: they may package scripts or command-line workflows, but they may also contain only background knowledge, constraints, procedures, examples, or other task-specific context (Zhang et al., 2025). They also differ in how they appear in context. Whereas tools are typically exposed through callable interfaces and argument schemas (Aizawa, 2025), skills use progressive disclosure: the agent initially sees only compact metadata, and full instructions, script arguments, API parameters, or implementation details become visible only after the skill is loaded (Anthropic, 2025, 2026).

Reusable experience and skills. A second line of work asks how agents can accumulate reusable knowledge. Voyager writes executable code skills in Minecraft and retrieves them for later tasks (Wang et al., 2023); Reflexion stores verbal feedback (Shinn et al., 2023); and ExpeL extracts reusable natural-language lessons from prior experience (Zhao et al., 2024). More recent systems make this reuse more explicit: ReUseIt synthesizes reusable web workflows (Liu et al., 2025b), SkillWeaver discovers and refines web-agent skills (Zheng et al., 2025), and SkillRL distills trajectories into hierarchical skill banks for reinforcement learning (Xia et al., 2026). We instead assume an existing skill library and study whether agents can route to the right skills as it grows.

Long-context use and retrieval. Retrieval-augmented generation retrieves evidence before generation (Lewis et al., 2020); dense passage retrieval provides scalable semantic search (Karpukhin et al., 2020); and generator-side studies such as Fusion-in-Decoder (FiD) examine how models consume retrieved passages (Izacard and Grave, 2021). HyDE and Self-RAG further show that retrieval can be improved by generated hypothetical documents or adaptive retrieval decisions (Gao et al., 2023; Asai et al., 2024). Studies of long-context utilization and needle-in-a-haystack retrieval show that placing relevant information in a long context does not guarantee that models use it (Liu et al., 2024a; Bai et al., 2024; Kamradt, 2023). Du et al. further show that context length alone can reduce performance even with perfect retrieval and even when irrelevant tokens are replaced or masked (Du et al., 2025). The results in Section 4 provide a contrasting mechanism for skill routing: length is not sufficient to explain the collapse, because replacing irrelevant skill metadata with placeholders largely restores golden-skill loading; the larger problem is semantic interference from plausible skill descriptions and instruction interference when routing is folded into execution.

7 Limitations

For budget reasons, the proprietary-model validation does not cover the full task set or every routing setting in the main study. We therefore treat it as a smaller consistency check on a 37-task subset, with the corresponding results reported in Appendix E.

8 Conclusion

We document skill routing collapse: as skill libraries grow, LLM agents increasingly fail to load relevant skills even when those skills remain available. Preselection can stabilize skill use, with tradeoffs that depend on library size. Controlled interventions show that the failure is driven by semantic and instruction interference, not context growth alone. Thus, large skill libraries need explicit routing and skill-level evaluation. By explaining this failure, our analysis also informs future skill-selection and skill-protocol design.

References

Ken Aizawa. 2025. Writing effective tools for AI agents—using AI agents. https://www.anthropic.com/engineering/writing-tools-for-agents. Accessed 2026-05-24.

Anthropic. 2025. Agent skills. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview. Claude Platform Docs; accessed 2026-05-24.

Anthropic. 2026. Extend claude with skills. https://code.claude.com/docs/en/skills. Accessed 2026-05-06.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the International Conference on Learning Representations*.

Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, Yanxu Chen, Yicun Chen, Yimin Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Ziwei Chen, Dazhi Cheng, Minghan Chu, Jialei Cui, Jiaqi Deng, Muxi Diao, Hao Ding, Mengfan Dong, Mengnan Dong, Yuxin Dong, Yuhao Dong, Angang Du, Chenzhuang Du, Dikang Du, Lingxiao Du, Yulun Du, Yu Fan, Shengjun Fang, Qiulin Feng, Yichen Feng, Garimugai Fu, Kelin Fu, Hongcheng Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Chengyang Gong, Xiaochen Gong, Zhuoma Gongque, Qizheng Gu, Xinran Gu, Yicheng Gu, Longyu Guan, Yuanying Guo, Xiaoru Hao, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Jiaxi Hu, Yangyang Hu, Zhenxing Hu, Ke Huang, Ruiyuan Huang, Weixiao Huang, Zhiqi Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yu Jing, Guokun Lai, Aidi Li, C. Li, Cheng Li, Fang Li, Guanghe Li, Guanyu Li, Haitao Li, Haoyang Li, Jia Li, Jingwei Li, Junxiong Li, Lincan Li, Mo Li, Weihong Li, Wentao Li, Xinhang Li, Xinhao Li, Yang Li, Yanhao Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zheming Li, Weilong Liao, Jiawei Lin, Xiaohan Lin, Zhishan Lin, Zichao Lin, Cheng Liu, Chenyu Liu, Hongzhang Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Tianyu Liu, Weizhou Liu, Xiangyan Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yuanxin Liu, Yue Liu, Zhengying Liu, Zhongnuo Liu, Enzhe Lu, Haoyu Lu, Zhiyuan Lu, Junyu Luo, Tongxu Luo, Yashuo Luo, Long Ma, Yingwei Ma, Shaoguang Mao, Yuan Mei, Xin Men, Fanqing Meng, Zhiyong Meng, Yibo Miao, Minqing Ni, Kun Ouyang, Siyuan Pan, Bo Pang, Yuchao Qian, Ruoyu Qin, Zeyu Qin, Jiezhong Qiu, Bowen Qu, Zeyu Shang, Youbo Shao, Tianxiao Shen, Zhennan Shen, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Feifan Song, Pengwei Song, Tianhui Song, Xiaoxi Song, Hongjin Su, Jianlin Su, Zhaochen Su, Lin Sui, Jinsong Sun, Junyao Sun, Tongyu Sun, Flood Sung, Yunpeng Tai, Chuning Tang, Heyi Tang, Xiaojuan Tang, Zhengyang Tang, Jiawen Tao, Shiyuan Teng, Chaoran Tian, Pengfei Tian, Ao Wang, Bowen Wang, Chensi Wang, Chuang Wang, Congcong Wang, Dingkun Wang, Dinglu Wang, Dongliang Wang, Feng Wang, Hailong Wang, Haiming Wang, Hengzhi Wang, Huaqing Wang, Hui Wang, Jiahao Wang, Jinhong Wang, Jiuzheng Wang, Kaixin Wang, Linian Wang, Qibin Wang, Shengjie Wang, Shuyi Wang, Si Wang, Wei Wang, Xiaochen Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yipu Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhexu Wang, Zihan Wang, Zizhe Wang, Chu Wei, Ming Wei, Chuan Wen, Zichen Wen, Chengjie Wu, Haoning Wu, Junyan Wu, Rucong Wu, Wenhao Wu, Yuefeng Wu, Yuhao Wu, Yuxin Wu, Zijian Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Yuchong Xie, Yifei Xin, Bowei Xing, Boyu Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinbo Xu, Xinran Xu, Yangchuan Xu, Yichang Xu, Yuemeng Xu, Zelai Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Guangyao Yang, Hao Yang, Junwei Yang, Kai Yang, Ningyuan Yang, Ruihan Yang, Xiaofei Yang, Xinlong Yang, Ying Yang, Yi Yang, Yi Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Dan Ye, Wenjie Ye, Zhuorui Ye, Bohong Yin, Chengzhen Yu, Longhui Yu, Tao Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Xiaokun Yuan, Yang Yue, Weihao Zeng, Dunyuan Zha, Haobing Zhan, Dehao Zhang, Hao Zhang, Jin Zhang, Puqi Zhang, Qiao Zhang, Rui Zhang, Xiaobin Zhang, Y. Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yushun Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Chenguang Zhao, Feifan Zhao, Jinxiang Zhao, Shuai Zhao, Xiangyu Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Junfeng Zhong, Longguang Zhong, Weiming Zhong, M. Zhou, Runjie Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yuxuan Zhu, Zhen Zhu, Jingze Zhuang, Weiyu Zhuang, Ying Zou, and Xinxing Zu. 2026. Kimi K2.5: Visual agentic intelligence. *Preprint*, arXiv:2602.02276.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Babu Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. 2025. Context length alone hurts LLM performance despite perfect retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23281–23298, Suzhou, China. Association for Computational Linguistics.

Tiantian Gan and Qiyao Sun. 2025. RAG-MCP:

Mitigating prompt bloat in LLM tool selection via retrieval-augmented generation. *Preprint*, arXiv:2505.03275.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. MetaTool benchmark for large language models: Deciding whether to use tools and which to use. In *Proceedings of the International Conference on Learning Representations*.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*.

Greg Kamradt. 2023. Needle in a haystack: Pressure testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. 2022. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *Preprint*, arXiv:2205.00445.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

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

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee. 2026. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *Preprint*, arXiv:2602.12670.

George Ling, Shanshan Zhong, and Richard Huang. 2026. Agent skills: A data-driven analysis of claude skills for extending large language model functionality. *Preprint*, arXiv:2602.08004.

Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Syed Fahad Allam Shah, and Dan Roth. 2025a. ToolScope: Enhancing LLM agent tool use through tool merging and context-aware filtering. *Preprint*, arXiv:2510.20036. ACL Main Conference 2026.

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

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024b. AgentBench: Evaluating LLMs as agents. In *Proceedings of the International Conference on Learning Representations*.

Yimeng Liu, Misha Sra, Jeevana Priya Inala, and Chenglong Wang. 2025b. ReUseIt: Synthesizing reusable AI agent workflows for web automation. *Preprint*, arXiv:2510.14308.

MiniMax AI. 2026. MiniMax M2.5: Built for real-world productivity. https://www.minimax.io/news/minimax-m25. Official release page.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*.

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. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and

Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *Preprint*, arXiv:2305.16291.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *Preprint*, arXiv:2602.08234.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations*.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. 2026. GLM-5: From vibe coding to agentic engineering. *Preprint*, arXiv:2602.15763.

Barry Zhang, Keith Lazuka, and Mahesh Murag. 2025. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Engineering at Anthropic; accessed 2026-05-24.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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

A SkillsBench Dataset Details

SkillsBench is a benchmark for evaluating how agent skills affect LLM-agent task solving (Li et al., 2026). The original paper describes a community-driven construction process in which 105 contributors submitted 322 candidate tasks, followed by automated checks and human review. It states that the benchmark contains 86 tasks, excludes two GPU-dependent tasks from evaluation, and evaluates 84 tasks across 11 domains. Each task is a self-contained module with a human-written instruction, a containerized environment, a reference solution, deterministic verifier scripts, and one or more curated golden skills.

We use the official SkillsBench GitHub repository. The repository's task directory contains 87 task directories, so our counts use the repository contents as the source of truth. We exclude the five tasks in Appendix B for infrastructure or environment reliability reasons, leaving 82 adopted tasks in the main controlled study. Table 8 reports how many golden skills these adopted tasks contain. Most tasks require only a small number of skills, but the benchmark still includes multi-skill workflows that test whether routing can recover several complementary procedures.

$\|G\|$	Tasks
1	26
2	20
3	16
4	11
5	6
6	2
7	1

Table 8: Distribution of golden-skill counts among the 82 adopted SkillsBench tasks. $|G|$ is the number of golden skills for a task.

Table 9 gives the average resource footprint of one valid implicit-routing run as the number of added irrelevant skills increases. The table averages over the adopted tasks and the three open-weight execution models, using the same valid-run filter as the main token-cost analysis. Even before adding irrelevant skills, a single run averages more than fifteen minutes and roughly half a million input tokens, which is why the full study is expensive and why the 1,000-skill diagnostic interventions use a smaller task subset.

s	Runs	Time	Input toks	Output toks
0	730	15.3m	485k	13k
10	731	16.7m	481k	13k
100	728	15.4m	619k	13k
1000	727	17.0m	1.95M	13k

Table 9: Average valid-run resource footprint under implicit routing by number of added irrelevant skills. Time is wall-clock execution time per run; token columns are LLM input and output tokens per run.

A.1 License compliance

The SkillsBench repository includes an Apache License 2.0 license file. We do not modify or redistribute the benchmark; we use it as an evaluation benchmark and report the resulting experimental measurements. We preserve attribution through citation to the original paper. Our use is therefore consistent with the permissions and notice requirements of Apache License 2.0.

A.2 Server configuration

The experiments were executed on Linux servers with 32 logical CPU cores and 62.3 GiB of system memory. The CPU model is AMD EPYC 7K62 48-Core Processor, configured as one socket with 16 physical cores and two hardware threads per core.

B Main Study Task Exclusions

The main controlled study excludes the five tasks listed in Table 10 because we encountered infrastructure or environment problems that prevented reliable execution.

#	Task
1	fix-visual-stability
2	mhc-layer-impl
3	multilingual-video-dubbing
4	python-scala-translation
5	speaker-diarization-subtitles

Table 10: Tasks excluded from the main controlled study for infrastructure or environment reasons.

C Experimental Subset Tasks

For budget reasons, the controlled interventions in Section 4 use the 10-task subset listed in Table 11. The same task list is used for the original, masked, plan-first, and explicit-preselection comparisons in that section.

#	Task
1	adaptive-cruise-control
2	data-to-d3
3	earthquake-phase-association
4	econ-detrending-correlation
5	enterprise-information-search
6	manufacturing-equipment-maintenance
7	react-performance-debugging
8	reserves-at-risk-calc
9	simpo-code-reproduction
10	weighted-gdp-calc

Table 11: Tasks used in the 10-task experimental subset.

D Full Data for Scaling Curves

The main text uses Figure 3 and Figure 4 to emphasize the shape of the scaling effects. This appendix gives the exact values behind those curves so that the visual trends can be checked without reading points from the plots. The full model-by-strategy Pass@3 matrix is already reported in Table 2; the additional tables here focus on the plotted slices and on the skill-loading measurements that are not otherwise visible in full.

D.1 Implicit-routing scaling by execution model

Section 3.2 argues that implicit routing breaks down before final task success fully reflects the failure. Tables 12–14 provide the three panels of Figure 3 in tabular form. Table 12 reports end-to-end Pass@3 for the implicit-routing condition, Table 13 reports the fraction of golden skills actually loaded during execution, and Table 14 reports the share of runs that load no skill at all.

The comparison across these tables shows the masking effect discussed in the main text. Pass@3 declines for all three execution models by the 1,000-skill setting, but the routing metrics deteriorate earlier and more sharply. For example, GLM-5 keeps Pass@3 near its small-library value at 100 irrelevant skills, while its loaded-skill recall has already dropped substantially and its zero-load rate has increased. This is the empirical reason we treat loaded-skill recall and zero-load rate as first-class metrics rather than relying only on verifier success.

D.2 Skill scaling by routing strategy

Section 3.3 compares implicit routing with three explicit-routing strategies. Tables 15–17 give the values behind Figure 4: aggregate Pass@3, loaded-skill recall, and zero-load rate for each routing strategy and library size. These tables are complementary to Table 2: the main-text table breaks task success down by execution model, whereas this appendix keeps the strategy-level view and adds the two skill-use metrics.

Model	0	10	100	1k
GLM-5	57.3	59.8	56.1	42.7
Kimi K2.5	52.4	56.8	50.6	43.9
MiniMax M2.5	45.1	46.3	41.5	29.3

Table 12: Values for Figure 3(a): implicit-routing Pass@3 (%) by execution model and number of added irrelevant skills.

Model	0	10	100	1k
GLM-5	79.6	71.8	55.0	20.9
Kimi K2.5	78.8	74.4	63.2	40.5
MiniMax M2.5	62.3	58.2	41.6	21.5

Table 13: Values for Figure 3(b): implicit-routing loaded-skill recall (%) by execution model.

E Budget-Limited Proprietary Model Validation

The main experiments use open-weight models to keep evaluation cost manageable. This appendix asks whether the same implicit-routing pressure is visible for stronger proprietary frontier models. Because these runs are substantially more expensive, we evaluate a smaller 37-task subset and treat the result as a qualitative consistency check rather than a second full benchmark.

Table 18 lists the exact proprietary-validation subset.

Table 19 reports Pass@3 on that shared subset for three proprietary models and, for comparison, the three open-weight models evaluated on the same tasks. Table 20 then reports loaded-skill recall for the proprietary models.

The proprietary models show the same qualitative pressure from larger skill libraries. Their task success is generally more stable than the open-weight models on this subset: Claude Opus 4.6 remains near 60 Pass@3 even at 1,000 irrelevant skills.

However, Table 20 makes the routing degradation clearer. Claude Opus 4.6 drops from 81.3 loaded-skill recall with no irrelevant skills to 30.9 at 1,000; Gemini 3.1 Pro drops from 77.4 to 43.0; and GPT-5.4 drops from 76.2 to 49.3. The decline is sometimes slower than in the open-weight main study, especially for Gemini and GPT, but the direction is the same. Relevant skills remain

Model	0	10	100	1k
GLM-5	7.8	9.4	26.0	64.9
Kimi K2.5	7.0	7.9	16.7	31.3
MiniMax M2.5	13.1	20.7	39.3	58.4

Table 14: Values for Figure 3(c): implicit-routing zero-load rate (%) by execution model. Lower is better.

Strategy	0	10	100	1k	10k
Naive	51.6	54.3	49.4	38.6	–
LLM preselect	54.9	52.4	54.9	51.6	–
RAG	52.4	48.0	49.6	46.5	41.5
Hybrid preselection	52.8	53.7	50.0	52.8	50.4

Table 15: Values for Figure 4(a): aggregate Pass@3 (%) by routing strategy and number of added irrelevant skills. "–" indicates omitted infeasible settings.

Strategy	0	10	100	1k	10k
Naive	73.6	68.2	53.3	27.4	–
LLM preselect	74.0	73.6	73.0	62.0	–
RAG	74.6	70.1	69.3	58.6	48.5
Hybrid preselection	74.0	72.9	70.5	67.1	55.2

Table 16: Values for Figure 4(b): loaded-skill recall (%) by routing strategy.

Strategy	0	10	100	1k	10k
Naive	9.3	12.7	27.3	51.6	–
LLM preselect	8.5	8.4	9.5	9.8	–
RAG	8.1	11.7	12.3	13.2	18.0
Hybrid preselection	8.1	8.9	10.7	9.1	10.7

Table 17: Values for Figure 4(c): zero-load rate (%) by routing strategy. Lower is better.

installed, yet implicit execution-time routing becomes less reliable as the library grows. This also support the same conclusion as the main experiment: stronger final task performance does not imply that implicit routing remains healthy.

F Precision Metrics

This appendix reports precision metrics for runtime skill loading and preselection. Tables 21 and 22 report loaded-skill precision for open-weight and proprietary models, respectively; Table 23 reports preselected-skill precision. As the number of skills grows, agents often load fewer skills or no skills at all, so loaded precision usually declines less sharply than recall. Loaded precision is computed over runs with at least one loaded skill; selected precision is computed over nonempty preselected sets.

G Skill Demand Breakdown

The main results aggregate over tasks with different numbers of golden skills. This section checks whether the scaling story is driven only by tasks with many required skills or whether it also appears for simpler tasks. Table 24 groups tasks by the number of golden skills in the task environment: single-skill tasks, tasks with two or three golden skills, and tasks with four or more golden skills. Within each library size s, the table reports both Pass@3 and loaded-skill recall for Naive exposure, LLM preselection, RAG, and hybrid preselection.

The table supports the same interpretation as the main text while adding a task-demand perspective. At 1,000 irrelevant skills, Naive exposure drops for both single-skill and multi-skill groups, showing that routing collapse is not limited to the hardest tasks. Loaded-skill recall shows the mechanism more directly: with 1,000 irrelevant skills, Naive exposure loads far fewer golden skills than the explicit-routing strategies across all three demand bins. Multi-skill tasks nevertheless make the mechanism more consequential: partial retrieval can find one useful skill but still miss another workflow needed for a passing solution. This is why the paper emphasizes all-gold coverage, loaded-skill recall, and trace analysis in addition to Pass@3.

H Hybrid Candidate Aggregation Details

Using multiple queries raises a design question: how should the retrieved lists be merged into one candidate pool? Table 25 compares three intuitive aggregators. `Max` keeps the best similarity score that any query gives a skill. `Mean` averages scores across queries. `Quota` takes a fixed number of top unseen skills from each query, encouraging coverage of different sub-procedures. Quota is the default in this paper, and Algorithm 2 in Appendix I gives the corresponding procedure. Quota gives the strongest all-gold coverage at 10,000 irrelevant skills, which is important for multi-skill tasks where missing one required workflow can break execution.

I Execution and Routing Algorithms

The main text summarizes the execution protocol; this appendix spells out the steps. Algorithm 1 receives a task, a skill set, a preselector, and an execution model. Line 1 applies the preselector to decide which skills are exposed. Line 2 formats the exposed skill metadata, and Line 3 builds the

Task	Task
3d-scan-calc	flink-query
adaptive-cruise-control	flood-risk-analysis
azure-bgp-oscillation-route-leak	gh-repo-analytics
citation-check	glm-lake-mendota
dynamic-object-aware-egomotion	gravitational-wave-detection
earthquake-phase-association	grid-dispatch-operator
earthquake-plate-calculation	hvac-control
econ-detrending-correlation	invoice-fraud-detection
energy-ac-optimal-power-flow	jax-computing-basics
energy-market-pricing	jpg-ocr-stat
enterprise-information-search	lab-unit-harmonization
exceltable-in-ppt	lake-warming-attribution
exoplanet-detection-period	latex-formula-extraction
financial-modeling-qa	lean4-proof
find-topk-similiar-chemicals	manufacturing-codebook-normalization
fix-build-agentops	manufacturing-equipment-maintenance
fix-build-google-auto	manufacturing-fjsp-optimization
fix-druid-loophole-cve	mario-coin-counting
fix-erlang-ssh-cve	

Table 18: Tasks used in the proprietary-model validation subset.

Model	0	10	100	1k
Claude Opus 4.6†	64.9	62.2	59.5	62.2
Gemini 3.1 Pro†	59.5	51.4	48.6	51.4
GPT-5.4†	54.1	56.8	43.2	43.2
GLM-5	54.1	62.2	54.1	35.1
Kimi K2.5	48.6	58.3	50.0	45.9
MiniMax M2.5	35.1	45.9	37.8	21.6

Table 19: Pass@3 (%) on the 37-task proprietary-model validation subset. Numeric columns give the number of added irrelevant skills on top of each task's golden skills. Models marked with † are proprietary; the remaining rows are the three open-weight execution models evaluated on the same task subset under implicit routing.

initial model context. Lines 4–7 repeat model tool calls and append tool outputs to the context. Lines 8–9 run the verifier and return its result.

Algorithm 2 gives the hybrid preselection procedure. Line 1 generates multiple search queries from the task instruction. Line 2 embeds the skill metadata, and Line 3 initializes an empty candidate set. Lines 4–8 retrieve top unseen candidates for each generated query and merge them into one candidate pool. Lines 9–10 ask the final selector to choose from that candidate pool and return the selected skills.

Algorithm 1 Skill-conditioned agent execution

Input: task $\tau = (I, F, V, G)$, skills S_s, preselector π, execution model M_e
Output: verifier result r
1 $P \leftarrow \pi(I, S_s)$
2 $M \leftarrow$ FormatMetadata(P)
3 $x \leftarrow$ BuildContext(I, M)
4 **while** HasToolCall$(y \leftarrow M_e(x))$ **do**
5 $\quad o \leftarrow$ Execute(y, F)
6 $\quad x \leftarrow$ Append(x, y, o)
7 **end while**
8 $r \leftarrow V(F)$
9 **return** r

Model	0	10	100	1k
Claude Opus 4.6†	81.3	77.7	43.3	30.9
Gemini 3.1 Pro†	77.4	70.4	57.9	43.0
GPT-5.4†	76.2	62.9	51.8	49.3

Table 20: Loaded-skill recall (%) on the 37-task proprietary-model validation subset. The statistic counts golden skills whose full instructions are accessed during execution.

Model	0	10	100	1k	10k
Naive					
GLM-5	99.6	99.5	97.1	88.6	–
Kimi K2.5	99.6	99.0	97.7	87.2	–
MiniMax M2.5	99.7	99.1	95.2	76.4	–
LLM preselect					
GLM-5	99.6	99.6	97.7	90.8	–
Kimi K2.5	99.5	99.4	96.6	84.9	–
MiniMax M2.5	99.7	99.5	97.2	78.6	–
RAG					
GLM-5	99.7	99.2	99.2	91.9	83.6
Kimi K2.5	99.7	99.2	99.5	91.0	80.7
MiniMax M2.5	99.7	98.7	98.8	90.2	75.3
Hybrid preselection					
GLM-5	99.4	99.3	97.7	90.7	82.6
Kimi K2.5	99.6	99.6	98.3	92.0	76.8
MiniMax M2.5	99.3	99.6	97.2	86.1	70.3

Table 21: Runtime loaded-skill precision (%) for open-weight models by routing strategy and number of added irrelevant skills.

Algorithm 2 Hybrid preselection

Input: instruction I, skills S_s, models M_p, M_{emb}, queries n, candidate budget B, final budget k
Output: selected skills P
1 $Q \leftarrow M_p$.GenerateQueries(I, n)
2 $E \leftarrow \{(u, M_{emb}(m(u))) : u \in S_s\}$
3 $C \leftarrow \emptyset$
4 **for** $q \in Q$ **do**
5 $\quad v_q \leftarrow M_{emb}(q)$
6 $\quad R_q \leftarrow$ RankBySim(v_q, E)
7 $\quad C \leftarrow C \cup$ TopUnseen$(R_q, \lceil B/n \rceil)$
8 **end for**
9 $P \leftarrow M_p$.SelectSkills(I, C, k)
10 **return** P

Algorithm 3 lists the three baseline preselection snippets. The Naive snippet returns the whole installed skill set. The LLM preselection snippet first formats the full metadata list, then asks a selector model to name skills, and finally matches the returned names back to installed skills. The RAG snippet embeds the instruction and skill metadata, ranks skills by embedding similarity, and returns the top-k skills.

Model	0	10	100	1k
Claude Opus 4.6†	100.0	100.0	100.0	95.2
Gemini 3.1 Pro†	100.0	100.0	100.0	100.0
GPT-5.4†	100.0	97.3	90.5	82.7

Table 22: Runtime loaded-skill precision (%) for proprietary models under implicit routing on the validation subset.

Method	0	10	100	1k	10k
LLM preselect	100.0	99.7	80.0	63.0	–
RAG	100.0	47.6	43.9	37.1	30.1
Hybrid preselection	100.0	99.8	84.5	72.0	52.6

Table 23: Preselected-skill precision (%) by method and number of added irrelevant skills.

Algorithm 3 Baseline preselectors

Input: instruction I, skills S_s, budget k, selector model M_p, embedding model M_{emb}
Output: exposed skill set P

Naive(I, S_s)
1 **return** S_s

LLMPreselect(I, S_s, k, M_p)
1 $M \leftarrow$ FormatMetadata(S_s)
2 $N \leftarrow M_p$.SelectSkills(I, M, k)
3 **return** MatchSkills(N, S_s)

RAGPreselect(I, S_s, k, M_{emb})
1 $v_I \leftarrow M_{emb}(I)$
2 $E \leftarrow \{(u, M_{emb}(m(u))) : u \in S_s\}$
3 $R \leftarrow$ RankBySim(v_I, E)
4 **return** TopK(R, k)

J Prompt Templates

This appendix gives the prompt templates. The execution agent receives skill metadata and must explicitly load a skill before using its longer instructions; preselection prompts instead return only skill names so that selected skills can be matched against the golden set.

J.1 Execution-agent initial prompt skeleton

Execution-agent initial prompt skeleton

```
You are an AI assistant tasked with solving command−line tasks in a
    Linux environment. You will be given a task description and the
    output from previously executed commands. Your goal is to solve
    the task by providing batches of shell commands.

Format your response as JSON with the following structure:

{
  "analysis": "Analyze the current state based on the terminal output
      provided. What do you see? What has been accomplished? What
      still needs to be done?",
  "plan": "Describe your plan for the next steps. What commands will
      you run and why? Be specific about what you expect each
      command to accomplish.",
  "commands": [
    {"keystrokes": "ls −la\n", "duration": 0.1},
    {"keystrokes": "cd project\n", "duration": 0.1}
  ],
  "task_complete": true
}

Required fields:
− "analysis": Your analysis of the current situation
− "plan": Your plan for the next steps
− "commands": Array of command objects to execute

Optional fields:
− "task_complete": Boolean indicating if the task is complete (defaults to
    false if not present)

Command object structure:
− "keystrokes": String containing the exact keystrokes to send to the
    terminal (required)
− "duration": Number of seconds to wait for the command to complete (
    defaults to 1.0 if not present)

IMPORTANT: The text inside "keystrokes" will be used completely
    verbatim. Most bash commands should end with a newline (\n) to
    execute.

−−−
SKILL SYSTEM:
If skills are available (shown in "available_skills"), you MUST check if
    any skill matches the task BEFORE executing commands.

Skill context:
{skills_block}

To load a skill, respond with ONLY this JSON (no other fields):
{"load_skill": "skill−name"}

After a skill is loaded, proceed with the normal response format above.
Skills contain best−practice workflows − use them instead of writing
    custom solutions.
−−−

Task Description:
{instruction}

Current terminal state:
{terminal_state}
```

The execution prompt separates invocation from routing. It can only load skills that appear in the exposed metadata block, so the preselection stage directly determines the action space visible to the agent.

J.2 LLM preselection prompt

LLM preselection prompt

```
You are a skill−routing assistant.
Given a task description and available skills,
select the skills most likely to be useful.

Return your answer as a JSON array of skill names. Return at most {
    max_skills}
skills. If no skill is relevant, return an empty array: []

IMPORTANT: Return ONLY the JSON array.
No explanation. No markdown code fences.

## Available Skills
{skills_list}

## Task Description
{task_instruction}

## Your Selection (JSON array of skill names)
```

This prompt uses a strict JSON-only output contract. This makes selected-skill logging straightforward and prevents explanations from being accidentally treated as skill names.

s	$\|G\|$	Tasks	Naive		LLM		RAG		Hybrid	
			P3	Load	P3	Load	P3	Load	P3	Load
0	1	26	53.8	91.9	52.6	93.6	52.6	95.7	56.4	95.7
	2-3	36	52.8	68.7	55.6	68.1	55.6	68.7	50.0	67.2
	≥ 4	20	46.7	58.8	56.7	59.3	46.7	57.8	53.3	58.0
10	1	26	51.9	91.8	53.8	94.9	48.7	89.3	57.7	94.9
	2-3	36	58.3	60.9	52.8	67.6	49.1	63.3	53.7	66.6
	≥ 4	20	50.0	51.2	50.0	56.6	45.0	57.3	48.3	55.6
100	1	26	51.9	73.2	55.1	94.4	50.0	88.9	55.1	92.3
	2-3	36	49.1	47.5	57.4	65.7	51.9	61.7	45.4	63.1
	≥ 4	20	46.7	38.4	50.0	58.3	45.0	57.5	51.7	55.6
1000	1	26	38.5	38.0	52.6	79.5	42.3	72.0	55.1	89.3
	2-3	36	37.0	26.8	51.9	54.8	47.2	52.8	50.9	59.0
	≥ 4	20	41.7	14.8	50.0	52.4	50.9	51.5	53.3	53.0
10000	1	26	–	–	–	–	38.5	64.1	47.4	70.1
	2-3	36	–	–	–	–	43.5	42.4	54.6	48.3
	≥ 4	20	–	–	–	–	41.7	39.1	46.7	48.4

Table 24: Pass@3 and loaded-skill recall (%) by skill-pool size and number of golden skills. The row group s gives the number of added irrelevant skills; $|G|$ bins tasks by their number of golden skills. Load is loaded-skill recall.

s	Agg.	Rec	All	Any
1000	Max	97.9	95.4	100.0
1000	Mean	93.9	89.7	97.7
1000	Quota	96.3	92.0	98.9
10000	Max	88.7	80.5	95.4
10000	Mean	88.7	78.2	95.4
10000	Quota	91.8	86.2	97.7

Table 25: Hybrid candidate quality (%) at budget 100.

J.3 Hybrid preselection query-generation prompt

Hybrid preselection query-generation prompt

```
You are a retrieval query planner for a skill−selection system.
Given a task description, write {num_queries}
short search queries covering distinct
capabilities likely needed for the task.

Guidelines:
− Each query should be concise, concrete, and skill−oriented.
− Prefer capability/library/workflow phrases.
− Do not restate the whole task.
− Return ONLY a JSON array of strings.
− No explanation. No markdown.

## Task Description
{task_instruction}

## Search Queries (JSON array)
```

The query-generation step is useful for multi-skill tasks because the original task instruction may emphasize the final artifact rather than every intermediate capability.

J.4 RAG-to-LLM final selection prompt

RAG-to-LLM final selection prompt

```
You are a budget−aware skill−routing assistant.
Given a task and candidate skills from search,
select only truly useful skills.

Return your answer as a JSON array of skill names. Return at most {
    max_skills}
skills. Use fewer when only one or two are needed.
If none is relevant, return an empty array: []

IMPORTANT: Return ONLY the JSON array.
No explanation. No markdown code fences.

## Candidate Skills Retrieved by RAG
{skills_list}

## Task Description
{task_instruction}

## Your Selection (JSON array of skill names)
```

This final-selection prompt is deliberately budget-aware: the candidate pool may contain semantically related but redundant skills, and the execution agent is only given the final compact subset.

J.5 Intervention prompts

The masked intervention uses the same execution prompt as above, but the patched skill block keeps golden-skill metadata readable and replaces every irrelevant skill's name, description, and location with length-matched `<mask>` tokens. A representative masked prompt is:

Masked execution prompt

```
You are an AI assistant tasked with solving command−line tasks in a
    Linux environment. You will be given a task description and the
    output from previously executed commands. Your goal is to solve
    the task by providing batches of shell commands.

Format your response as JSON with the following structure:
{ ... normal JSON response schema ... }

−−−
SKILL SYSTEM:
If skills are available (shown in "available_skills"), you MUST check if
    any skill matches the task BEFORE executing commands.

Skill context:
available_skills:
[
  {
    "name": "pid−controller",
    "description": "Implement PID control loops for control systems.",
```

```
    "location": "/root/.terminus/skills/pid−controller/SKILL.md"
  },
  {
    "name": "<mask> <mask>",
    "description": "<mask> <mask> <mask> <mask> <mask> <mask>",
    "location": "<mask> <mask> <mask> <mask> <mask>"
  },
  {
    "name": "<mask>",
    "description": "<mask> <mask> <mask> <mask> <mask> <mask> <
        mask>",
    "location": "<mask> <mask> <mask> <mask>"
  },
  ...
]

To load a skill, use: cat <location> (where location is from the skill's <
        location> tag above)

LOADED SKILLS:
{active_skills}

To load a skill, respond with ONLY this JSON (no other fields):
{"load_skill": "skill−name"}

After a skill is loaded, proceed with the normal response format above.
Skills contain best−practice workflows − use them instead of writing
        custom solutions.
−−−

Task Description:
{instruction}

Current terminal state:
{terminal_state}
```

The plan-first intervention keeps the original skill metadata and inserts one additional instruction immediately before the task description:

Plan-first execution prompt

```
You are an AI assistant tasked with solving command−line tasks in a
        Linux environment. You will be given a task description and the
        output from previously executed commands. Your goal is to solve
         the task by providing batches of shell commands.

Format your response as JSON with the following structure:
{ ... normal JSON response schema ... }

−−−
SKILL SYSTEM:
If skills are available (shown in "available_skills"), you MUST check if
        any skill matches the task BEFORE executing commands.

Skill context:
{skills_block}

To load a skill, respond with ONLY this JSON (no other fields):
{"load_skill": "skill−name"}

After a skill is loaded, proceed with the normal response format above.
Skills contain best−practice workflows − use them instead of writing
        custom solutions.
−−−
IMPORTANT: Before executing any commands, you MUST first explicitly
        think about which of the available skills (if any) are relevant
        to this task. State your reasoning about skill relevance, then
        load the skills you need. Only after that should you begin
        solving the task.
Task Description:
{instruction}

Current terminal state:
{terminal_state}
```

K Use of AI Assistants

AI assistants were used during the preparation of this paper for writing support and for assistance with parts of the experimental code and analysis scripts. The authors reviewed and edited the resulting text and code.

L Per-task Scaling Heatmaps

The main text reports aggregate Pass@3. This appendix adds a task-level view of that metric. Figures 6–9 show per-task Pass@3 as the number of added irrelevant skills increases. Each row is one SkillsBench task, each column is a library-size setting, and the color encodes Pass@3 from red (all runs fail) to blue (all runs pass).

The heatmaps are sorted by the Naive drop from no irrelevant skills to 1,000 irrelevant skills, so collapses under implicit routing appear near the top. This ordering makes the aggregate result more concrete: some tasks remain stable, but many tasks show abrupt failures once the library reaches hundreds or thousands of irrelevant skills. The explicit-routing heatmaps then show whether those same rows become more stable when the execution agent sees a smaller selected skill set.

Figure 6 shows the most direct form of routing collapse: rows that are blue or light blue at small scale often turn light red or red as irrelevant skills are added. Figure 7 shows the complementary result from the main text: when the full metadata list still fits, explicit LLM preselection keeps many of those rows more stable. Figures 8 and 9 focus on the scalable retrieval-based setting. RAG remains feasible at 10,000 skills but produces more mixed task-level outcomes; hybrid preselection reduces some of this variance by constructing a more diverse candidate pool before final selection.

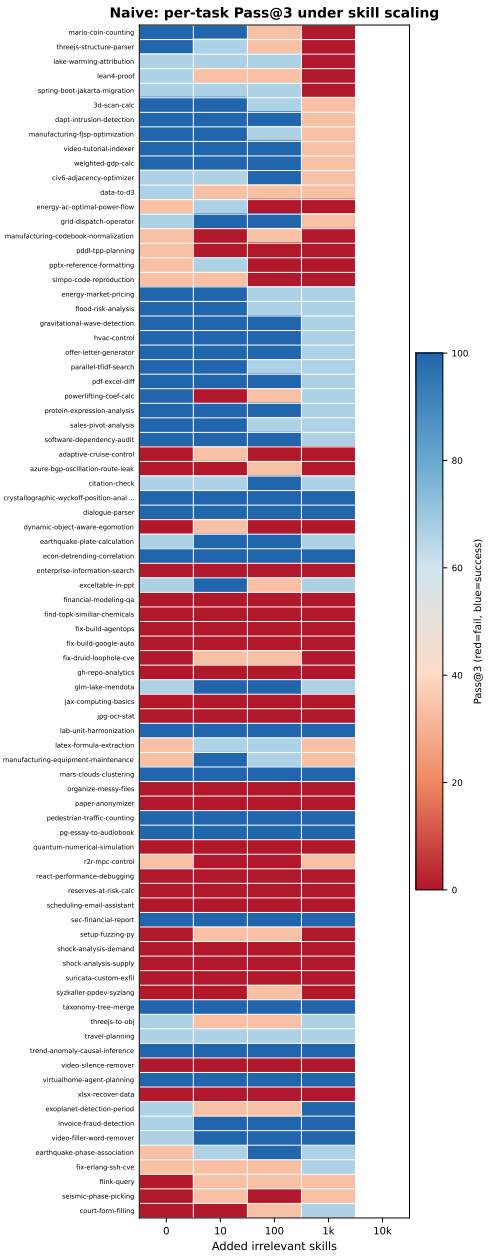

Figure 6: Per-task Pass@3 for Naive exposure as the number of added irrelevant skills grows. Red means all trials fail and blue means all trials pass. Rows are tasks, sorted by the naive drop from no irrelevant skills to 1,000 irrelevant skills.

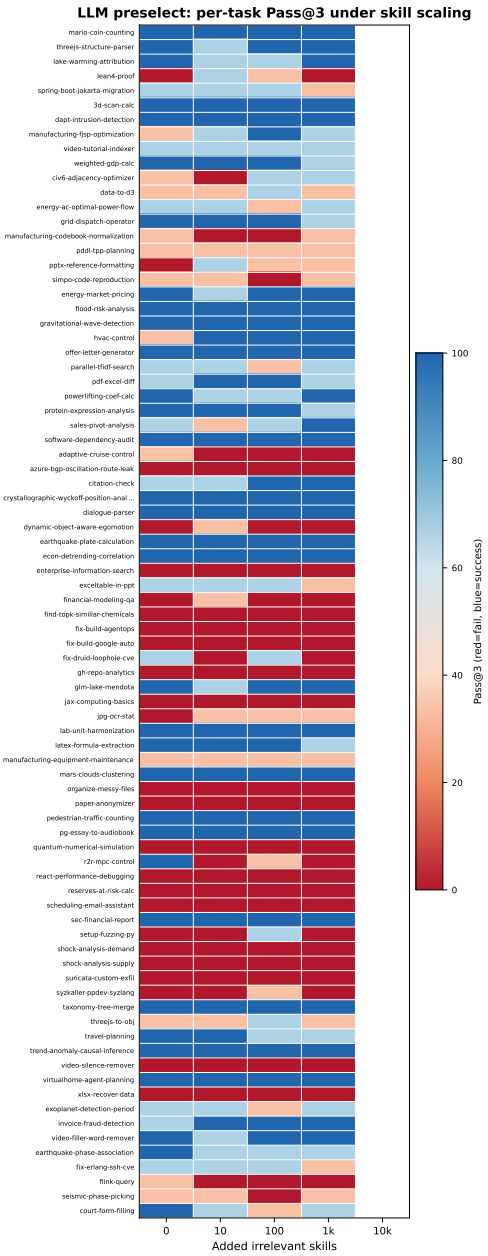

Figure 7: Per-task Pass@3 for LLM preselection. Compared with Naive exposure, many rows remain blue or light blue as the skill library grows, showing that explicit routing stabilizes task success when the full metadata list still fits in context.

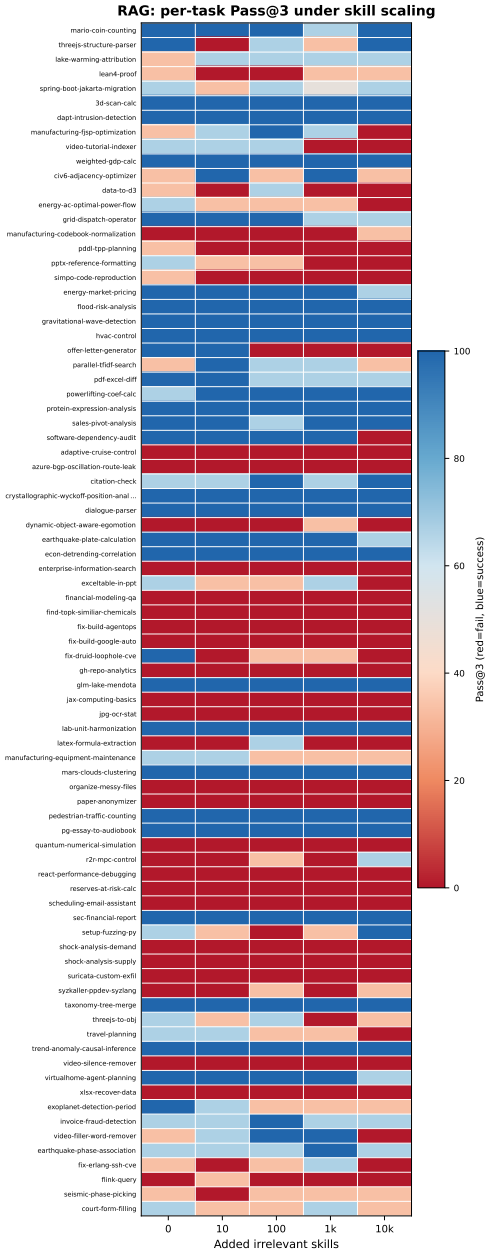

Figure 8: Per-task Pass@3 for embedding RAG. RAG is feasible at the 10k scale, but the mixed colors show that semantic retrieval alone misses some task-specific skill combinations.

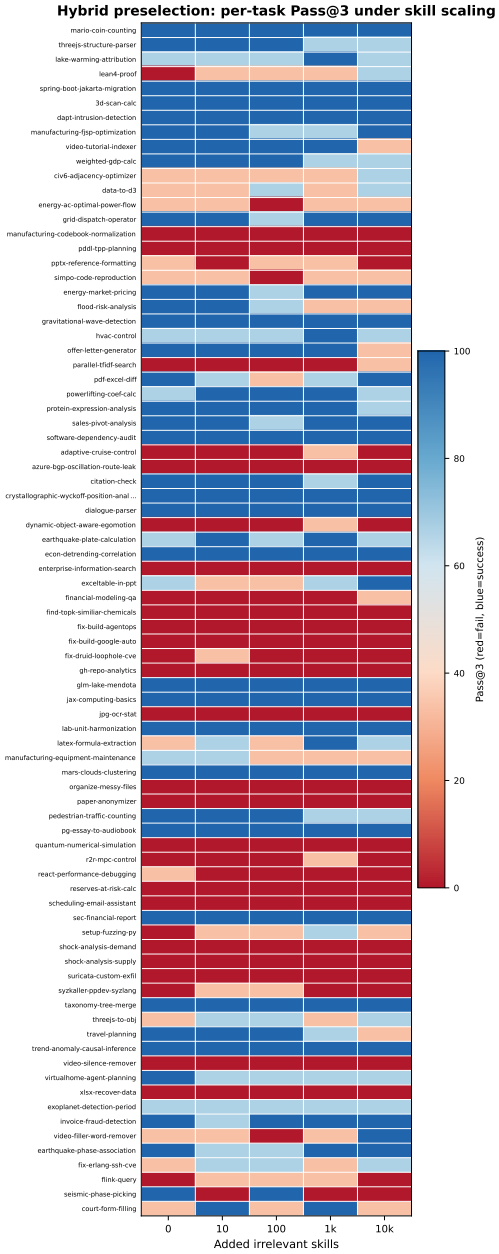

Figure 9: Per-task Pass@3 for hybrid preselection. The method keeps a compact candidate set while recovering more large-scale tasks than single-query RAG, especially when several capabilities must be retrieved together.