

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

Jinlin Liu¹ Wei Chen^{2*} Xiaojin Zhang^{2*}

¹The University of Hong Kong

²Huazhong University of Science and Technology

Abstract

Reusable skills let LLM agents load task-specific instructions on demand, but they make execution depend on implicit routing over skill metadata. We test whether this design scales by holding SkillsBench tasks and their golden skills fixed while adding irrelevant skills. We find *skill routing collapse*: agents increasingly load fewer or no skills, even though the relevant skills remain available. With 1,000 irrelevant skills, loaded-skill recall falls from 73.6% to 27.4% and the zero-load rate rises from 9.3% to 51.6%, whereas explicit selection from the same library retains 73.9% recall. Loading other skills, including plausible alternatives to golden skills, is too rare to explain this decline. Masking only the descriptions of irrelevant skills largely restores skill loading, yet with the metadata intact, explicitly asking agents to plan skill use still yields lower recall than preselection. These findings point to semantic interference from descriptions and instruction interference when skill choice competes with task execution. We also evaluate explicit preselection as a practical mitigation, including its coverage and latency trade-offs. Our analysis informs skill protocols and evaluation for agents with growing skill libraries.¹

1 Introduction

Large language model (LLM) agents now operate tools and files to complete multi-step workflows in software engineering, data analysis, document processing, and personal assistance (Yao et al., 2023; Liu et al., 2024b; Wang et al., 2024). Such agents need reusable procedures: a plotting routine, code-audit workflow, spreadsheet operation, or domain-specific calculation should not be rediscovered in every new task.

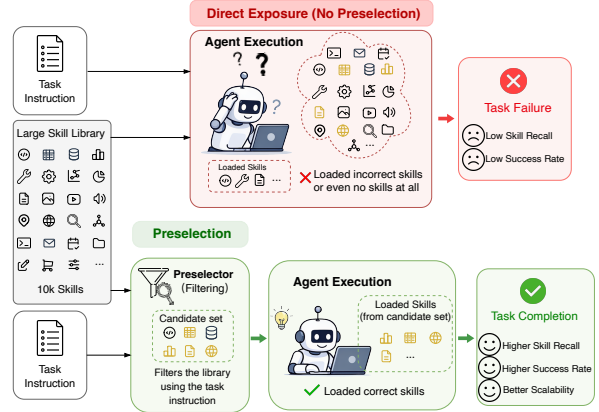


Figure 1: Lost in implicit routing. The relevant skills remain installed, but as irrelevant skills grow, agents increasingly fail to load them during implicit execution. Explicit routing recovers much of the missing skill recall.

Agent skills package such procedures and their supporting resources into reusable modules (Anthropic, 2025; Zhang et al., 2025), and public skill libraries are growing quickly: one analysis reports 2,179 listed skills on January 16, 2026 and 40,285 on February 5 (Ling et al., 2026). Skill selection is therefore becoming a scaling problem.

We use *skills* to mean Anthropic’s Agent Skills, whose progressive-disclosure design works as follows: the model first sees only skill names and descriptions, and opens a full skill file only if it judges the skill relevant (Anthropic, 2026). Routing is thus not a separate stage: the agent decides whether and which skills to load while planning actions, inspecting files, and solving the task. We call this execution-time selection *implicit routing*. It keeps long skill files out of context, but assumes that an agent can pick out the few relevant skills from many descriptions while simultaneously solving the task.

We test this assumption on SkillsBench (Li et al., 2026), which provides fixed task instructions, workspaces, verifiers, and relevant skills;

*Corresponding authors.

¹Code: <https://github.com/jinlinll/lost-in-implicit-routing>.

for each task, we keep these fixed and vary only the number of added irrelevant skills. The result is *skill routing collapse*: the right skills are installed, but agents increasingly load fewer skills or none at all. As irrelevant skills grow from 0 to 1,000, loaded-skill recall under implicit routing drops from 73.6% to 27.4% and the zero-load rate rises from 9.3% to 51.6%. Agents occasionally load non-golden alternatives, but too rarely to explain the collapse. Yet the same models retain 73.9% selected-skill recall when asked only to choose skills from the full library. Relevant skills thus remain recognizable; they are lost when routing becomes a subtask of execution.

End-to-end Pass@3 falls less sharply, from 51.6% to 38.6%, because strong agents can sometimes solve around missing skills; task success alone can therefore make a failing skill system look healthy. We make three contributions:

- We identify and measure *skill routing collapse*: growing libraries make agents load fewer or no skills, rather than substitute other skills for golden ones.
- We compare LLM, retrieval, and hybrid preselection on libraries of up to 10,000 added skills, and derive scale-dependent recommendations together with their token, coverage, and latency costs.
- We show that irrelevant descriptions suppress skill loading and that selecting skills while solving the task makes selection less reliable, revealing semantic and instruction interference beyond context growth alone.

2 A Controlled Test for Skill Routing

2.1 Task setup

We represent each task as $\tau = (I, F, V, G)$: instruction I , workspace F , executable verifier V , and the set G of relevant (golden) SkillsBench skills. A run succeeds only if V passes after the agent finishes; no LLM judge is used. With s added irrelevant skills, the installed library is $S_s = G \cup D_s$, where D_s is sampled independently of the task. Varying only s tests whether an agent can still find and load the needed skills as the metadata list grows (Figure 2).

2.2 Routing strategies

Under *implicit routing*, the executor sees the metadata of all skills in S_s and decides during execu-

tion whether to open any skill. *Explicit routing* adds a preselection step: a preselector receives the task and the metadata and exposes only a compact subset to the executor, which still follows progressive disclosure and must open a skill before using it. We compare three preselectors. *LLM preselection* chooses directly from the full metadata list; *RAG preselection* retrieves skills by embedding similarity; and *hybrid preselection* retrieves candidates and lets an LLM make the final choice (pseudocode in Appendix K). We also include a no-skill baseline.

Diagnosis versus mitigation. We use explicit routing in two ways. In *routing-only diagnosis*, the selector sees the task and the full library, including the same distractors as the executor, but only selects skills; selected-skill recall thus tests whether the needed skills are recognizable without task execution. In *preselection-plus-execution*, the executor then sees only the selected subset; loaded-skill recall and task success measure the practical benefit. The diagnosis changes only the objective, whereas the mitigation also narrows the executor’s library.

2.3 Metrics

Pass@3 is 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 opened during execution; the *zero-load rate* is the percentage of runs with $|L| = 0$, and *average skills loaded* is the mean of $|L|$. For explicit routing, *selected-skill recall* is $|G \cap P|/|G|$, where P is the preselected set. Appendix F reports precision, and Appendix I analyzes the loading of non-golden skills.

3 Main Findings

3.1 Experimental setup

SkillsBench tasks. SkillsBench (Li et al., 2026) provides task instructions, workspace files, executable verifiers, and task-specific golden skills. We exclude five tasks with infrastructure or environment problems and use the remaining 82 (Appendices A and B). Runs are costly: under implicit routing, one run averages 15.3 minutes and 485k input tokens without irrelevant skills, rising to 17.0 minutes and 1.95M input tokens with 1,000.

Skill libraries. We downloaded more than 20,000 skills from a skill market in leaderboard

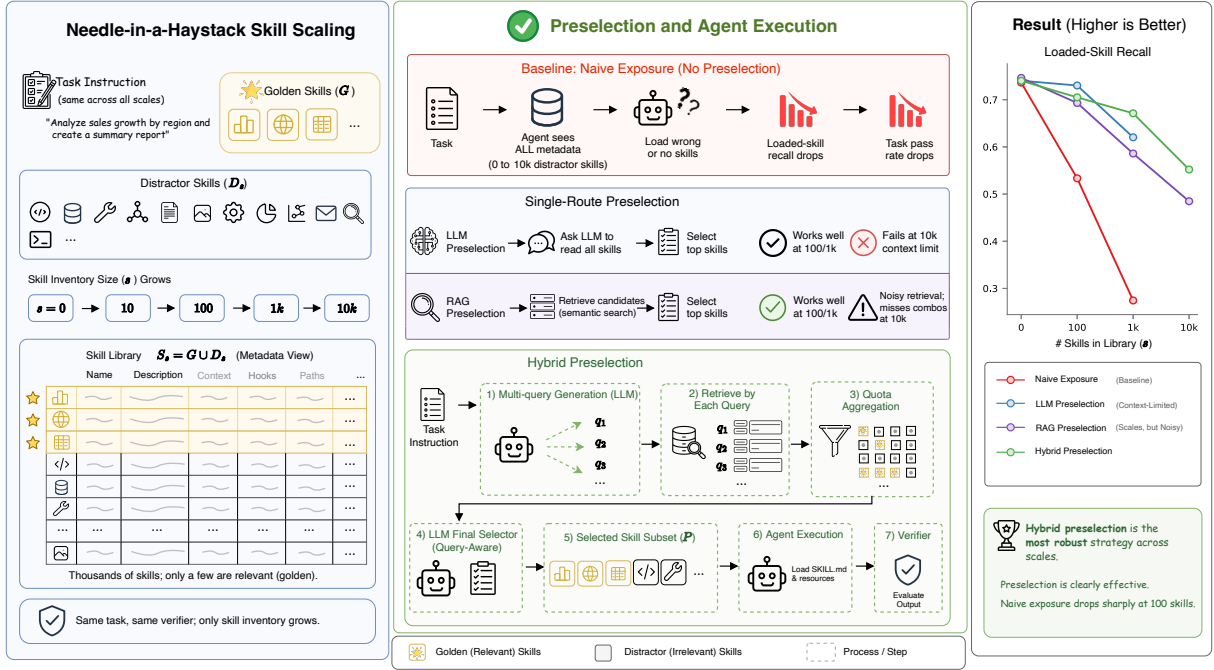


Figure 2: Overview of the controlled skill-scaling setup and routing strategies. Starting from an existing Skills-Bench task, we keep the instruction, files, verifier, and relevant skills fixed while varying only the number of added irrelevant skills.

order and sample irrelevant skills from this pool with a fixed seed (42), so the distractors are identical across models and strategies. We evaluate $s \in \{0, 10, 100, 1000, 10000\}$, always installing the golden skills. Every preselector exposes at most five skills: LLM-based selectors choose up to five, and RAG returns the top five by embedding similarity.

Models and scale. Given this cost, the main experiments use three open-weight executors: GLM-5 (Zeng et al., 2026), Kimi K2.5 (Bai et al., 2026), and MiniMax M2.5 (MiniMax AI, 2026); RAG and hybrid preselection use Qwen text-embedding-v4 embeddings. Each model runs every task and condition three times, and we average over the task-model-run grid, for more than 15,000 trajectories and 10 billion tokens in total. A budget-limited validation on Claude Opus 4.6, Gemini 3.1 Pro, and GPT-5.4 yields consistent conclusions (Appendix E).

3.2 Lost in Implicit Routing

3.2.1 Implicit loading degrades before task success

Figure 3 shows implicit routing for the three open-weight models. **Implicit routing collapses as irrelevant skills grow.** With only 100 irrelevant skills, loaded-skill recall already falls by 24.6, 15.6, and 20.7 points for GLM-5, Kimi K2.5,

and MiniMax M2.5 (Table 2), while zero-load rates rise: agents increasingly stop using skills at all rather than merely choosing wrong ones. Task success lags behind. At the same scale, GLM-5 keeps Pass@3 almost unchanged (57.3% to 56.1%) and Kimi K2.5 stays near its baseline (52.4% to 50.6%). Strong executors can solve around missing skills, so final success alone hides a failing skill protocol. Three proprietary models on a 37-task subset show the same pressure: Claude Opus 4.6, for example, keeps Pass@3 at 62.2% with 1,000 irrelevant skills (vs. 64.9%), yet its loaded-skill recall falls from 81.3% to 30.9% (Appendix E). The collapse is also not confined to multi-skill tasks: at 1,000 irrelevant skills, loaded-skill recall falls from 91.9% to 38.0% on single-skill tasks and from 58.8% to 14.8% on tasks with four or more golden skills (Appendix G). Appendix D.1 lists all plotted values.

3.2.2 Library growth suppresses skill use

Table 1 shows that from zero to 1,000 distractors, the average number of loaded skills falls from 1.72 to 0.66 and the zero-load rate rises from 9.2% to 50.8%. Could agents instead be replacing golden skills with other skills? Non-golden loading is rare (2.32% of evaluations), and only 0.95% of evaluations load a non-golden skill whose embedding cosine similarity to a missed golden skill exceeds

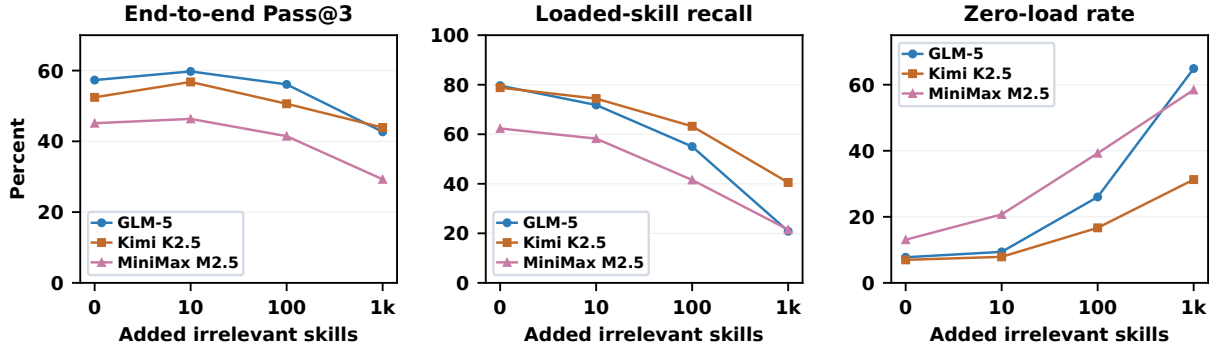


Figure 3: Implicit-routing scaling by execution model. (a) End-to-end Pass@3. (b) Loaded-skill recall. (c) Zero-load rate. Only the number of added irrelevant skills changes.

s	Avg. skills loaded	ZeroLoad	Load Rec.
0	1.72	9.2	73.5
10	1.54	13.0	67.8
100	1.20	27.9	52.9
1000	0.66	50.8	27.6

Table 1: Skill-loading behavior as the number of irrelevant skills grows. ZeroLoad and loaded-skill recall are percentages. This analysis covers 86 tasks, a slightly broader set than the main study (Appendix I).

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

0.8. Such loads concentrate at $s = 1000$ (7.19% and 3.79%, respectively), and the above-threshold cases are near-duplicates such as `pdf-reader` for `pdf` or `excel` for `xlsx`. When agents do load skills, the loads are mostly correct: under implicit routing at 1,000 irrelevant skills, loaded-skill precision is still 88.6%, 87.2%, and 76.4% for the three models (Appendix F). By construction, no distractor duplicates a golden skill. Routing collapse therefore primarily reflects under-loading, not substitution (details in Appendix I).

3.2.3 The same skills are recognizable when selected explicitly

Are the relevant skills intrinsically hard to recognize? Table 2 compares implicit loaded-skill recall with routing-only selected-skill recall for the same models and scales. **The same LLMs still identify the needed skills when selection is explicit.** At 1,000 irrelevant skills, GLM-5 loads 20.9% of golden skills but selects 82.5%, and Kimi K2.5 loads 40.5% but selects 80.7%; MiniMax M2.5 shows a smaller but still large gap (21.5% vs. 58.4%). The full metadata list thus often suffices for recognition when selection is the sole objective; the failure arises when selection is folded into reading files, planning commands, and completing the task.

3.3 Analyzing Explicit Routing

Since the bottleneck lies in routing during execution, we next ask how much preselection helps when the executor sees only the selected subset. We do not propose a new routing algorithm; we compare simple preselectors to show when this design helps and what it costs.

3.3.1 Explicit routing across library sizes

Figure 4 compares implicit routing with the three preselectors. **Narrowing the visible skill set reduces the scale effect:** Pass@3, loaded-skill recall, and zero-load rate all become less sensitive to library size. The best strategy depends on scale. LLM preselection is strongest around 100 added skills, whereas hybrid preselection is strongest at 1,000 and remains usable at 10,000, where full-list LLM preselection is omitted as infeasible. At 100 added skills, LLM preselection raises aggregate loaded-skill recall from 53.3% to 73.0% and Pass@3 from 49.4% to 54.9%; at 10,000, hybrid preselection reaches 55.2% recall and 50.4% Pass@3, versus 48.5% and 41.5% for RAG (Appendix D.2). Table 3 shows the same pattern for each executor: at 1,000 added skills, for instance, LLM preselection raises Pass@3 from 42.7% to 58.5% for GLM-5 and from 29.3% to 42.7% for MiniMax M2.5.

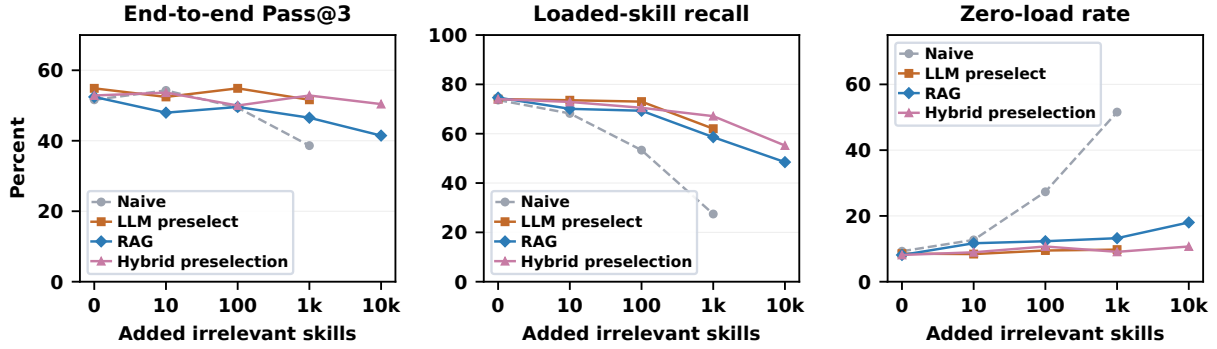


Figure 4: Skill scaling curves by routing strategy. (a) End-to-end Pass@3. (b) Loaded-skill recall. (c) Zero-load rate.

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 3: Per-model Pass@3 (%) by routing strategy and number of added irrelevant skills. The w/o column disables skills entirely; “–” marks omitted infeasible settings.

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

Table 4 reports selected-skill recall, which concerns the preselector’s choice rather than whether the executor later opens the skills, and should be read with selected precision (Table 25). RAG has the highest recall at 10 and 100 added skills, partly because it always returns five skills, but its precision is much lower (47.6% and 43.9%, vs. 99.8% and 84.5% for hybrid preselection). At large scale, hybrid preselection is preferable: it has the highest recall at 1,000 and improves over RAG at 10,000 (62.0 to 72.2) while also raising precision from 30.1% to 52.6%.

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

3.3.2 Token cost of routing

Table 5 separates LLM input and output tokens for execution and preselection; embedding cost is excluded because it is negligible (below \$1 in total). Preselection adds a routing call but can sharply shrink the execution context in large libraries. At 1,000 irrelevant skills, implicit routing consumes 1952k agent-input tokens, whereas LLM preselection spends 36k input tokens on routing and reduces agent input to 497k, for 547k in total. RAG is cheapest at 10,000 skills (469k) because it makes no LLM call, but it also performs worst at that scale (Figure 4). Hybrid preselection costs modestly more (547k, including 8k for preselection) and generalizes better.

3.3.3 Why hybrid preselection helps at 10k

A single RAG query built from the task instruction can over-focus on one part of a multi-step task and miss other required capabilities. Hybrid preselection instead has an LLM write several task-focused

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

queries, retrieves candidates for each, and lets an LLM select from the pooled candidates. Table 6 compares this pool with single-query RAG@100 before final selection: at 10,000 irrelevant skills, it raises candidate recall from 88.2 to 91.8 and all-gold coverage from 78.0 to 86.2, so needed skills are less often missing from the pool. To merge the per-query lists, we take a fixed quota of top unseen skills from each query, which gives higher all-gold coverage at 10,000 than max-score or mean-score merging (86.2 vs. 80.5 and 78.2; Appendix H).

Coverage and latency trade-offs. Fixed preselection can omit skills that become necessary later. As distractors grow from 10 to 1,000, the average number of golden skills exposed to the executor falls from 2.21 to 1.78 for LLM preselection, from 2.38 to 1.85 for RAG, and from 2.21 to 1.97 for hybrid preselection, compared with 2.52 when all golden skills are visible (Appendix J). Coverage alone does not determine success: at 1,000, RAG exposes more golden skills than LLM preselection but reaches lower Pass@3 (46.5% vs. 51.6%), suggesting that which skills survive selection also matters. Preselection also adds latency before execution even when it saves tokens: from 10 to 1,000 distractors, the median routing time rises from 8 to 19 seconds for LLM preselection and from 33 to 41 seconds for hybrid preselection, which makes two LLM calls (Table 31). RAG needs no LLM call; because we embedded all skills and task instructions in a batch before the experiments, its preselection reduces to a similarity lookup, so we do not report a separate latency for it. Preselection is thus a useful mitigation rather than a complete solution, and its benefit must be weighed against coverage loss and response time.

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

Table 7: Routing results (%) on the 10-task subset. All rows except the first use 1,000 irrelevant skills. Load Rec. is loaded-skill recall; Zero is the zero-load rate; All-gold is the share of runs that load or select every golden skill.

4 Why Agents Are Lost in Implicit Routing

4.1 Experimental setup

We use a 10-task subset at 1,000 irrelevant skills (Appendix C) and the three open-weight models, and test two interventions, each run three times per model. **Masked** keeps every distractor slot but replaces its name, description, and location with `<mask>` placeholders of approximately the original length. **Plan-first prompt** keeps the metadata intact but instructs the agent to consider skills before execution. Table 7 adds the no-extra-skills reference and preselection-plus-execution on the same subset. Field-level variants mask only names, descriptions, locations, or names plus locations; each runs once per task and model and is compared with a single run of the original condition, with full masking as the three-run reference. In every variant, golden-skill metadata stays readable (prompts in Appendix L.5).

4.2 Results and discussion

Skill metadata creates semantic interference. In Table 7, masking keeps the context long, yet largely restores no-extra-skills loading (63.9% vs. 73.5% recall), cuts the zero-load rate from 62.2% to 23.6%, and is competitive with the strongest preselection row (61.1%). Because the slots and approximate length are preserved, the gain is not an artifact of a shorter context. Long metadata alone therefore does not cause routing collapse; the failure appears when distractors carry meaningful descriptions that look like plausible alternative actions.

Descriptions dominate the field-level effect. Masking only descriptions raises loaded-skill recall to 63.3%, close to full masking (63.9%; Ta-

Setting	Avg. loaded	ZeroLoad	Load Rec.
Original metadata	0.50	60.0	18.3
Mask names	0.93	40.0	43.7
Mask descriptions	1.17	23.3	63.3
Mask locations	0.90	36.7	47.5
Mask names + locations	1.10	36.7	51.7
Mask all distractor (reference)	1.26	23.6	63.9

Table 8: Field-level masking of distractor metadata on the 10-task subset. Avg. loaded is the mean number of skills loaded; ZeroLoad and Load Rec. are percentages. The first five rows use one run per task and model; full masking is the three-run reference.

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

Table 9: 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.

ble 8). Masking names or locations alone helps less (43.7% and 47.5%), and masking both together reaches only 51.7%. The latter removes the explicit targets for `load_skill` and direct file reads, although agents could still find the files by exploring. That the remaining descriptions still suppress golden-skill loading shows that their effect extends beyond competition among explicit loading targets.

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, Table 9 compares the skills planned in the agent’s first response with isolated LLM preselection; this planned recall is lower than final loaded recall because agents sometimes load skills absent from the initial plan. Both settings see nearly the same context, the task instruction plus the full metadata, and differ mainly in the system prompt: LLM preselection asks only for skill selection, whereas Plan-first asks the agent to solve the task and to choose skills first. Despite this explicit reminder, Plan-first recalls far fewer golden skills (39.9% vs. 70.6%) and covers all golden skills in far fewer runs (20.5% vs. 54.4%). Routing and beginning the task thus compete within the same response.

Descriptions have two functional roles. These results suggest a link between the two forms of interference. During execution, descriptions act as conditional sub-instructions (load this skill when

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 counts both `load_skill` calls and direct reads of skill files.

its situation arises), creating a multi-instruction-following problem alongside the task. In routing-only selection, they are instead semantic-matching inputs rather than instructions to monitor. This role shift helps explain why agents select skills more reliably when selection is their sole objective, even with the full library visible.

4.3 Case study

Traces further support instruction interference. Figure 5 shows two plan-first failures. In Case 1, GLM-5 explicitly plans to load *pid-controller*, *vehicle-dynamics*, and *simulation-metrics*, but opens only the first. In Case 2, MiniMax M2.5 writes a step-by-step plan for a D3 visualization that mentions no skill, and never loads *d3-visualization*. Both runs fail.

A prompt nudge thus does not reliably separate routing from execution; reducing the visible skill set addresses both forms of interference more directly.

5 Practical Recommendations

Because irrelevant descriptions interfere with both routing and task execution, narrowing the visible skill set before execution is more reliable than strengthening prompt instructions, and the right method depends on scale. For small curated libraries, implicit routing remains usable, but systems should track which skills are selected and loaded rather than rely on task success alone. Around the hundred-skill scale, explicit preselection should become the default; direct LLM preselection is simple and strong while the full metadata list fits in context. For thousands of skills, full-list LLM preselection becomes expensive or

infeasible, and hybrid preselection (retrieval for a compact candidate set, then an LLM selector) is the strongest option in our experiments, improving relevant-skill coverage while keeping the execution context small. Since fixed preselection can omit skills needed later and adds latency, deployments should also monitor coverage and response time. Prompt reminders to consider skills are a lightweight safeguard that should complement, not replace, preselection. For evaluation, benchmarks of skill-using agents should report loaded-skill recall and zero-load rate alongside task success, since Pass@3 alone can hide routing failure.

6 Related Work

LLM agents and tool choice. ReAct couples reasoning with actions (Yao et al., 2023), Toolformer learns tool calls (Schick et al., 2023), and MRKL combines language models with external knowledge and reasoning modules (Karpas et al., 2022). As action spaces grow, Gorilla and Tool-LLM study 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 retrieves tools to counter prompt bloat (Gan and Sun, 2025). Overlapping tool names and descriptions also cause selection ambiguity, motivating 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 may also contain only background knowledge, constraints, procedures, or examples (Zhang et al., 2025). They also enter context differently. Tools are typically exposed through callable interfaces and argument schemas (Aizawa, 2025), whereas skills use progressive disclosure: the agent first sees compact metadata, and full instructions, script arguments, or implementation details become visible only after loading (Anthropic, 2025, 2026). The loading decision must therefore be made from metadata alone.

Reusable experience and skills. Voyager writes and retrieves executable code skills in Minecraft (Wang et al., 2023), Reflexion stores verbal feedback (Shinn et al., 2023), and ExpeL extracts natural-language lessons from prior experience (Zhao et al., 2024). More recently, ReUseIt

synthesizes reusable web workflows (Liu et al., 2025b), SkillWeaver discovers and refines web-agent skills (Zheng et al., 2025), and SkillIRL distills trajectories into hierarchical skill banks for reinforcement learning (Xia et al., 2026). These works focus on acquiring skills; we assume an existing library and ask whether agents can still 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 enables scalable semantic search (Karpukhin et al., 2020), and Fusion-in-Decoder studies how generators consume retrieved passages (Izacard and Grave, 2021); HyDE and Self-RAG improve retrieval with hypothetical documents or adaptive retrieval decisions (Gao et al., 2023; Asai et al., 2024). Long-context and needle-in-a-haystack studies show that models do not always use relevant information placed in context (Liu et al., 2024a; Bai et al., 2024; Kamradt, 2023), and Du et al. (2025) find that context length alone can hurt performance even with perfect retrieval and masked irrelevant tokens. Skill routing behaves differently: replacing irrelevant metadata with placeholders largely restores golden-skill loading, so the collapse stems mainly from semantic and instruction interference rather than length alone (Section 4).

7 Conclusion

We documented skill routing collapse: as libraries grow, agents increasingly load fewer or no skills even though the relevant skills remain available, and loading other skills instead is too rare to explain this decline. Masking only irrelevant descriptions largely restores skill loading, whereas explicitly planning skill use with the metadata intact remains less effective than preselection. These findings point to semantic interference from descriptions and instruction interference when skill choice competes with task completion. Preselection can stabilize skill use but introduces coverage and latency trade-offs. Growing skill libraries therefore call for explicit routing and for evaluating skill use, not task success alone. A natural next step is routing that revisits the selected subset during execution, recovering skills that fixed preselection omits without exposing the full library.

Limitations

For budget reasons, the proprietary-model validation covers only a 37-task subset and not every routing setting of the main study, so we treat it as a consistency check (Appendix E).

Field-level masking uses a small subset and fewer repetitions than full masking, so numerical proximity between conditions does not establish equivalence. Name-plus-location masking hides identifiers in the prompt but does not delete skill files. Embedding similarity screens plausible alternatives, not functional interchangeability. The two roles we attribute to descriptions are an interpretation of behavioral evidence, not a measurement of internal model processing. Routing latency depends on API serving conditions and does not establish end-to-end speedup; because RAG used embeddings computed in advance, our setup does not capture the cost of embedding new tasks online. Fixed preselection can omit skills needed later (Appendix J).

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, Qulin 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, Jiayi 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, Xinhao 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, Mingqing Ni, Kun Ouyang, Siyuan Pan, Bo Pang, Yuchao Qian, Ruoyu Qin, Zeyu Qin, Jiezhong Qiu, Bowen Qu, Zeyu Shang, Youbo Shao, Tianxiao Shen, Zhenan 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, Shiyan 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. [Meta-Tool 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 Tenenholz. 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. [Agent-Bench: 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, Shangdong 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, Yuhao 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, which contains 87 tasks; our counts follow the repository. We exclude the five tasks in Appendix B for infrastructure or environment reliability reasons, leaving 82 adopted tasks in the main controlled study. Table 10 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 10: Distribution of golden-skill counts among the 82 adopted SkillsBench tasks. $|G|$ is the number of golden skills for a task.

Table 11 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. 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.

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 evalu-

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 11: 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.

ation 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 main experiments were executed on Linux servers with an AMD EPYC 7K62 processor (16 physical cores, 32 hardware threads) and 62.3 GiB of memory. The field-level masking experiments ran in containerized Linux environments on an Apple Silicon machine with 16 GB of memory; they measure skill-loading behavior and are not used for latency comparisons. LLM inference was provided through Tencent TokenPlan.

B Main Study Task Exclusions

The main controlled study excludes the five tasks listed in Table 12 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 12: 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 13. The same task list is used for the original, masked, plan-first, and explicit-preselection comparisons in that section.

D Full Data for Scaling Curves

The main text uses Figure 3 and Figure 4 to emphasize the shape of the scaling effects. This ap-

#	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 13: Tasks used in the 10-task experimental subset.

pendix 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 3; 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 14–16 provide the three panels of Figure 3 in tabular form. Table 14 reports end-to-end Pass@3 for the implicit-routing condition, Table 15 reports the fraction of golden skills actually loaded during execution, and Table 16 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.

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 14: 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 15: Values for Figure 3(b): implicit-routing loaded-skill recall (%) by execution model.

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

D.2 Skill scaling by routing strategy

Section 3.3 compares implicit routing with three explicit-routing strategies. Tables 17–19 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 3: 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.

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 17: 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 18: 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 19: Values for Figure 4(c): zero-load rate (%) by routing strategy. Lower is better.

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 20 lists the exact proprietary-validation subset.

Task	Task
3d-scan-calc	fink-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-similar-chemicals	manufacturing-codebook-normalization
fix-build-agentops	manufacturing-equipment-maintenance
fix-build-google-auto	manufacturing-fjsp-optimization
fix-druid-loop-hole-cve	mario-coin-counting
fix-erlang-ssh-cve	

Table 20: Tasks used in the proprietary-model validation subset.

Table 21 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 22 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 22 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 installed, yet implicit execution-time routing becomes less reliable as the library grows. This supports the conclusion of the main experiment: stronger final task performance does not imply that implicit routing remains healthy.

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 21: 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.

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

F Precision Metrics

This appendix reports precision metrics for run-time skill loading and preselection. Tables 23 and 24 report loaded-skill precision for open-weight and proprietary models, respectively; Table 25 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 26 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.

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 23: Runtime loaded-skill precision (%) for open-weight models by routing strategy and number of added irrelevant 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 24: 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 25: Preselected-skill precision (%) by method and number of added irrelevant skills.

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 27 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 K 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 Non-Golden Skill Loading

Experimental scope. We examine skill loading for GLM-5, Kimi K2.5, and MiniMax M2.5 at $s \in \{0, 10, 100, 1000\}$ across all three runs. This analysis covers 86 tasks, whereas the main task-success study uses the adopted 82-task subset. Table 1 therefore summarizes a slightly broader task set than the main-study curves.

A skill counts as loaded if the agent calls `load_skill` on it or reads its `SKILL.md` file; references to skills outside the exposed library do not count. Each task’s golden set is restricted to golden skills present in the library.

Similarity matching. For each evaluation, let $N = L \setminus G$ be the loaded non-golden skills and $M = G \setminus L$ the missed golden skills. We compare every pair in $N \times M$, using the RAG embedding model on the full skill name and description, formatted as `name: description`. An evaluation counts as loading a potential alternative when any pair has cosine similarity strictly greater than 0.8. We neither match across tasks nor compare against golden skills already loaded. The threshold screens semantic overlap; it does not certify functional equivalence.

Alternative loading is uncommon. Non-golden loads occur in 2.32% of evaluations across scales, and potential alternatives above the threshold occur in 0.95%. They concentrate at $s = 1000$ (Table 28) and include clear semantic neighbors such as `pdf-reader` and `pdf`, or `excel` and `xlsx`. Table 29 lists all above-threshold task-model-skill-pair groups, combining repeated runs. Although non-golden loading increases at the largest scale, it remains far less common than the decline in skill use. Thus, routing collapse primarily reflects under-loading, not substitution of other

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 26: 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 27: Hybrid candidate quality (%) at budget 100.

s	Non-golden load rate	> 0.8 alternative rate
0	0.00	0.00
10	0.26	0.00
100	1.83	0.00
1000	7.19	3.79

Table 28: Non-golden skill loading (%). Non-golden load rate measures whether any non-golden skill is loaded; alternative rate additionally requires similarity above 0.8 to a missed golden skill in the same evaluation.

skills for golden skills.

Exact duplicates. Distractors are sampled so that none shares its identifier with a golden skill of the task; exact duplicates of golden skills therefore never appear as distractors. Differently named near-duplicates can remain and are covered by the similarity analysis above.

J Preselection Trade-offs and Latency

Fixed coverage. Preselection reduces the executor’s visible library but can remove a skill that becomes relevant later. Table 30 reports the average number of golden skills exposed to the executor, computed from the intersection of the selected set with task golden skills and averaged over task–

model identities. Task success uses the main-study Pass@3 values. Each preselection method retains fewer golden skills as distractors grow. Reduced coverage limits task success because omitted skills are unavailable during execution, although success also depends on which skills are missed and the quality of the remaining set. For example, RAG can retain more golden skills than LLM preselection yet yield lower task success; aggregate coverage alone does not determine whether the most consequential skills survive selection.

Routing-stage latency. We measure routing-stage latency on the adopted main-study tasks with the three open-weight models across all three runs. Routing-stage latency is the time from the start of preselection until the selected metadata is passed to the executor. For hybrid preselection, it covers query generation, query embedding and retrieval, and final LLM selection, but not the one-time construction of the skill embedding index. Environment setup, task execution, and verification are excluded.

The median delay increases from 8 to 19 seconds for LLM preselection and from 33 to 41 seconds for Hybrid as distractors grow from 10 to 1,000. Hybrid preselection makes two LLM calls (query generation and final selection), whereas LLM preselection makes one. Because the two methods were run at different times, these numbers reflect the API serving conditions of each period and are not a controlled latency comparison.

We do not report routing latency for RAG. It makes no generative selector call, and we embedded all skill metadata and task instructions in a

Task	Model	Runs	Loaded non-golden → missed golden	Cosine
court-form-filling	Kimi K2.5	2	pdf-generator → pdf	0.922
financial-modeling-qa	Kimi K2.5	1, 2, 3	pdf-reader → pdf	0.903
financial-modeling-qa	MiniMax M2.5	1, 2	pdf-reader → pdf	0.903
invoice-fraud-detection	MiniMax M2.5	1, 3	pdf-reader → pdf	0.903
latex-formula-extraction	GLM-5	3	pdf-reader → pdf	0.903
latex-formula-extraction	MiniMax M2.5	1	pdf-reader → marker	0.826
latex-formula-extraction	MiniMax M2.5	1	pdf-reader → pdf	0.903
pdf-excel-diff	GLM-5	1	pdf-reader → pdf	0.903
pdf-excel-diff	Kimi K2.5	2	pdf-reader → pdf	0.903
pdf-excel-diff	MiniMax M2.5	2	pdf-reader → pdf	0.903
pg-essay-to-audiobook	GLM-5	2, 3	tts → gtts	0.811
protein-expression-analysis	Kimi K2.5	1	excel → xlsx	0.906
reserves-at-risk-calc	Kimi K2.5	1	excel → xlsx	0.906
sales-pivot-analysis	Kimi K2.5	2	excel → xlsx	0.887
sales-pivot-analysis	Kimi K2.5	3	pdf-reader → pdf	0.888
sales-pivot-analysis	MiniMax M2.5	3	excel → xlsx	0.887
shock-analysis-demand	Kimi K2.5	1	excel → xlsx	0.906
shock-analysis-demand	MiniMax M2.5	2, 3	excel → xlsx	0.906
shock-analysis-supply	Kimi K2.5	2	excel → xlsx	0.906
shock-analysis-supply	MiniMax M2.5	2	excel → xlsx	0.906
video-tutorial-indexer	GLM-5	1	transcriber → speech-to-text	0.843
weighted-gdp-calc	Kimi K2.5	1, 2, 3	excel → xlsx	0.906

Table 29: All above-threshold skill-pair groups for the three open-weight models, all at $s = 1000$. Runs lists which of the three repeated runs contain the pair. Similarities use full name-and-description embeddings; the same skill name can have task-specific descriptions.

Setting	Avg. golden exposed	Pass@3 (%)
All golden, no preselection	2.52	51.6
Naive-10	2.52	54.3
RAG-10	2.38	48.0
LLM-10	2.21	52.4
Hybrid-10	2.21	53.7
RAG-100	2.21	49.6
LLM-100	2.11	54.9
Hybrid-100	2.10	50.0
RAG-1000	1.85	46.5
LLM-1000	1.78	51.6
Hybrid-1000	1.97	52.8

Table 30: Fixed preselection trades reduced interference for missed golden-skill coverage. Method suffixes denote the number of added irrelevant skills.

Method	s	Mean (s)	Median (s)	P90 (s)
LLM	10	11.7	8.0	26
LLM	100	18.0	9.5	45
LLM	1000	35.4	19.0	73
Hybrid	10	38.9	33.0	75
Hybrid	100	43.7	37.0	89
Hybrid	1000	48.3	41.0	93

Table 31: Observed routing-stage latency before task execution. P90 is the 90th percentile. The two methods were measured in separate experiment periods, so differences also reflect serving conditions.

single batch before running the experiments, so RAG preselection at run time reduces to a similarity lookup over precomputed vectors. Embedding calls are typically fast relative to LLM generation, but in a deployment that embeds each new task online, this step would add a small cost that

our setup does not capture. Likewise, token savings need not translate into shorter end-to-end wall time, which also depends on task behavior, API latency, and environment operations. Practical deployments should weigh coverage and task success against the added routing delay, especially for interactive tasks.

K 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 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

Require: task $\tau = (I, F, V, G)$, skills S_s , preselector π , execution model M_e

Ensure: verifier result r

```
1:  $P \leftarrow \pi(I, S_s)$ 
2:  $M \leftarrow \text{FormatMetadata}(P)$ 
3:  $x \leftarrow \text{BuildContext}(I, M)$ 
4: while HasToolCall( $y \leftarrow M_e(x)$ ) do
5:    $o \leftarrow \text{Execute}(y, F)$ 
6:    $x \leftarrow \text{Append}(x, y, o)$ 
7: end while
8:  $r \leftarrow V(F)$ 
9: return  $r$ 
```

Algorithm 2 Hybrid preselection

Require: instruction I , skills S_s , models M_p, M_{emb} , queries n , candidate budget B , final budget k

Ensure: selected skills P

```
1:  $Q \leftarrow M_p.\text{GenerateQueries}(I, n)$ 
2:  $E \leftarrow \{(u, M_{emb}(m(u))) : u \in S_s\}$ 
3:  $C \leftarrow \emptyset$ 
4: for  $q \in Q$  do
5:    $v_q \leftarrow M_{emb}(q)$ 
6:    $R_q \leftarrow \text{RankBySim}(v_q, E)$ 
7:    $C \leftarrow C \cup \text{TopUnseen}(R_q, \lceil B/n \rceil)$ 
8: end for
9:  $P \leftarrow M_p.\text{SelectSkills}(I, C, k)$ 
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.

Algorithm 3 Baseline preselectors

Require: instruction I , skills S_s , budget k , selector model M_p , embedding model M_{emb}

Ensure: exposed skill set P

```
1: Naive( $I, S_s$ ):
2:   return  $S_s$ 
3: LLMPreselect( $I, S_s, k, M_p$ ):
4:    $M \leftarrow \text{FormatMetadata}(S_s)$ 
5:    $N \leftarrow M_p.\text{SelectSkills}(I, M, k)$ 
6:   return  $\text{MatchSkills}(N, S_s)$ 
7: RAGPreselect( $I, S_s, k, M_{emb}$ ):
8:    $v_I \leftarrow M_{emb}(I)$ 
9:    $E \leftarrow \{(u, M_{emb}(m(u))) : u \in S_s\}$ 
10:   $R \leftarrow \text{RankBySim}(v_I, E)$ 
11:  return  $\text{TopK}(R, k)$ 
```

L 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.

L.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 ($\backslash n$) 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, 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 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.

L.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 format, so selected skills can be matched unambiguously and explanations are never mistaken for skill names.

L.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.

L.4 Hybrid preselection LLM final selection prompt

RAG-to-LLM final selection 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 final-selection prompt is the same as the LLM preselection prompt.

L.5 Intervention prompts

The field-level variants use the same prompt and replace only the designated distractor fields: name, description, location, or name plus location. Unmasked fields and all golden-skill metadata remain readable. The number of placeholders is proportional to the original field's character length, approximating rather than exactly matching its token length. The field-level comparison is a targeted diagnostic rather than a test of statistical equivalence.

The masked intervention uses the same execution prompt as above, but the skill block keeps golden-skill metadata readable and replaces every irrelevant skill's name, description, and location with <mask> placeholders of approximately the original length. 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> <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, 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.

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}

M 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.

N 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).

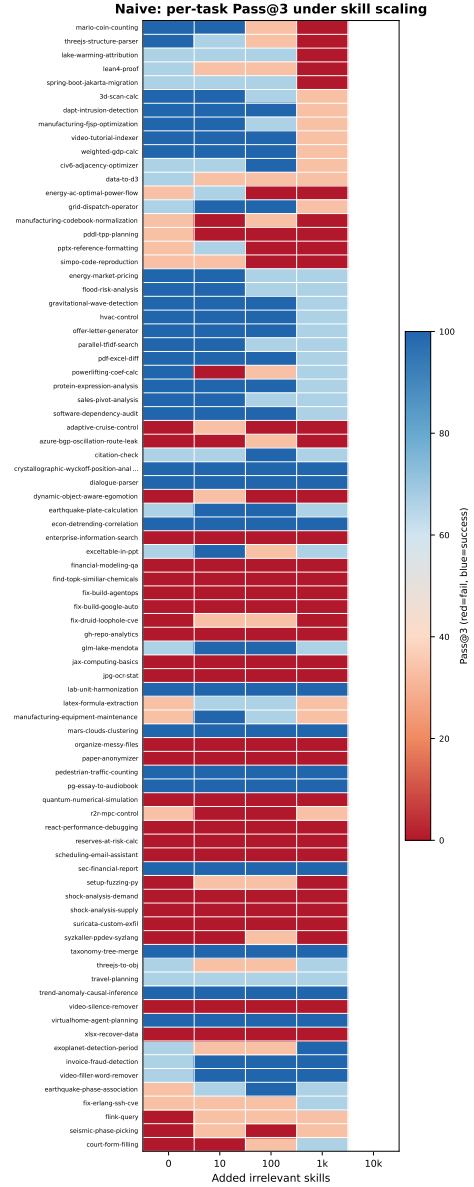


Figure 6: Per-task Pass@3 for Naive exposure as the number of added irrelevant skills grows. Red means all trials fail and blue means all trials pass. Rows are tasks, sorted by the naive drop from no irrelevant skills to 1,000 irrelevant skills.

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

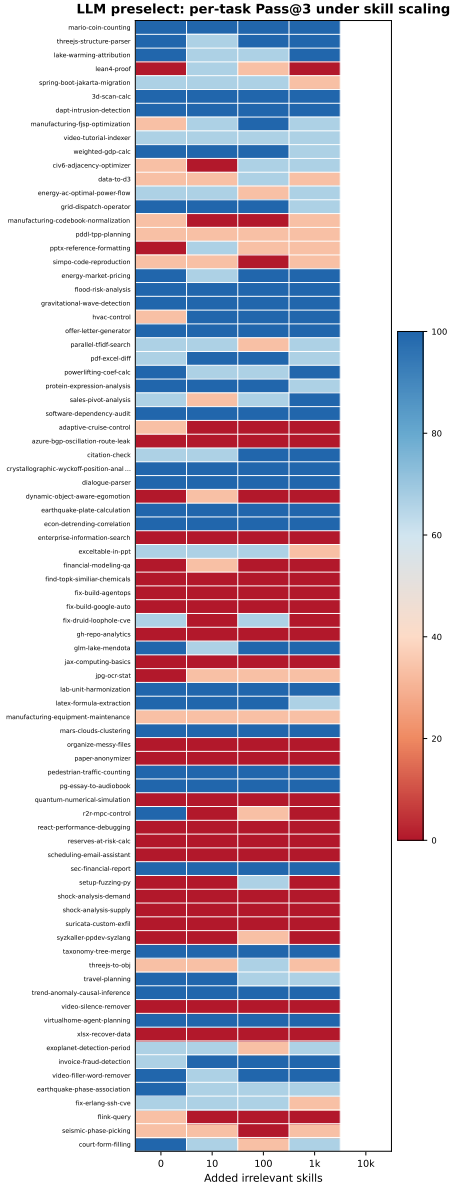


Figure 7: Per-task Pass@3 for LLM preselection. Compared with Naive exposure, many rows remain blue or light blue as the skill library grows, showing that explicit routing stabilizes task success when the full metadata list still fits in context.

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.

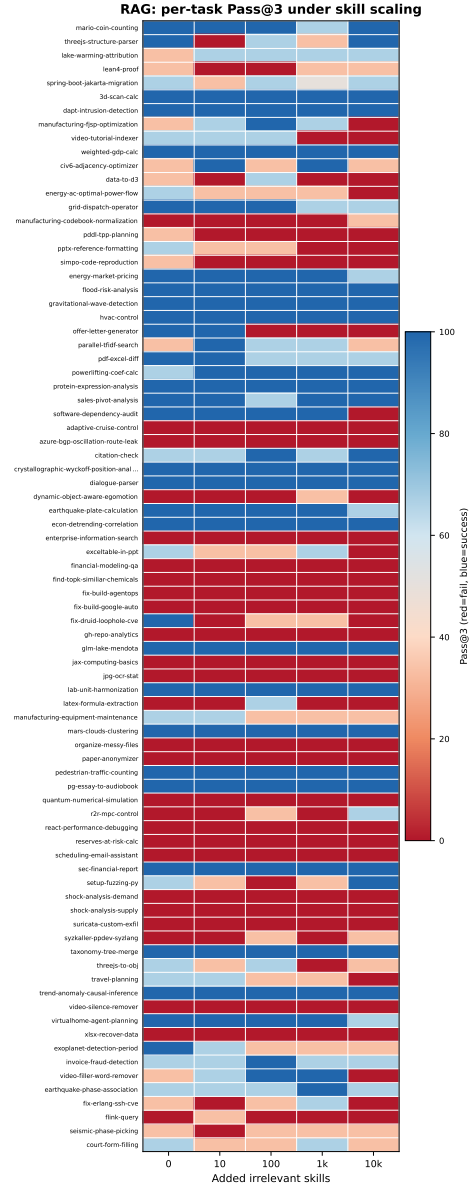


Figure 8: Per-task Pass@3 for embedding RAG. RAG is feasible at the 10k scale, but the mixed colors show that semantic retrieval alone misses some task-specific skill combinations.

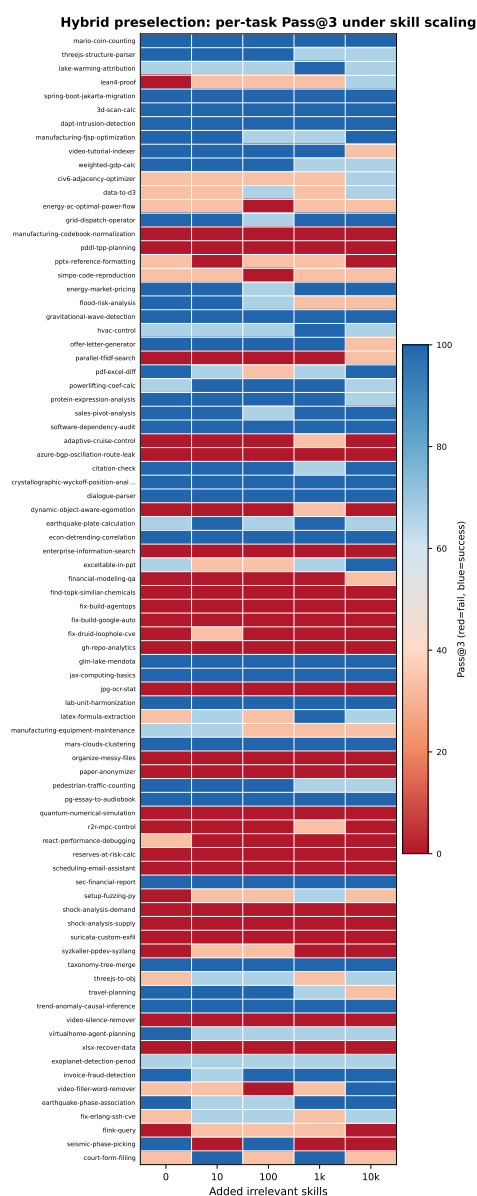


Figure 9: Per-task Pass@3 for hybrid preselection. The method keeps a compact candidate set while recovering more large-scale tasks than single-query RAG, especially when several capabilities must be retrieved together.