SkillFlow: Scalable and Efficient Agent Skill Retrieval System

Fangzhou Li[1,2,*], **Pagkratios Tagkopoulos**[1,2,3,*], **Ilias Tagkopoulos**[1,2,†]
[1]University of California, Davis,
[2]USDA/NSF AI Institute for Next Generation Food Systems,
[3]Process Integration and Predictive Analytics

arXiv:2504.06188v3 [cs.AI] 25 Sep 2026

Abstract

AI agents can extend their capabilities at inference time by loading reusable skills into context, yet equipping an agent with too many skills, particularly irrelevant ones, degrades performance. As community-driven skill repositories grow, agents need a way to selectively retrieve only the most relevant skills from a large library. We present *SkillFlow*, the first open, multi-stage retrieval system for agent skill discovery that frames skill acquisition as an information retrieval problem over a corpus of ∼35K community-contributed SKILL.md definitions indexed from GitHub. The pipeline progressively narrows a large candidate set through four stages (dense retrieval, two rounds of cross-encoder reranking, and LLM-based selection), balancing recall and precision at each stage. We evaluate SkillFlow on two coding benchmarks: SkillsBench, a benchmark of 87 tasks and 229 matched skills; and Terminal-Bench, a benchmark that provides only 89 tasks, and no matched skills. On SkillsBench, SkillFlow-retrieved skills raise Pass@1 from 9.2% to 16.4% (+78.3%, $p_{\text{adj}} = 3.64 \times 10^{-2}$), reaching 84.1% of the oracle ceiling, while on Terminal-Bench, agents readily use the retrieved skills (70.1% use rate) yet show no performance gain, revealing that retrieval alone is insufficient when the corpus lacks high-quality, executable skills for the target domain. SkillFlow demonstrates that framing skill acquisition as an information retrieval task is an effective strategy, and that the practical impact of skill-augmented agents hinges on corpus coverage and skill quality, particularly the density of runnable code and bundled artifacts. (https://github.com/IBPA/skill-flow)

1 Introduction

AI agents (systems that perceive their environment and act autonomously to achieve specific goals (Wooldridge & Jennings, 1995; Russell & Norvig, 2020; Franklin & Graesser, 1996)) have been a long-standing pursuit, from ancient automata (Price, 1964) to the intelligent agent programs of the 1990s and today's large language model (LLM) based agents (Durante et al., 2024). LLMs have dramatically expanded what agents can do, enabling them to reason over complex environments, generate and execute code, and solve multi-step tasks that were previously out of reach.

Despite these advances, agents today are typically equipped with a fixed set of capabilities, and the question of how they acquire and integrate new ones remains largely open. Agent Skills, introduced by Anthropic (Anthropic, 2025), handles this by defining a standardized format (SKILL.md) for encoding reusable procedural knowledge. The convention has been widely adopted by the open-source community, with a rapidly growing number of skill definitions shared across repositories on GitHub.

However, agents must load skills into their context at inference time, and equipping them with too many skills, particularly irrelevant ones, degrades both accuracy and efficiency (Li

∗ Equal contribution.
† Correspondence to: itagkopoulos@ucdavis.edu

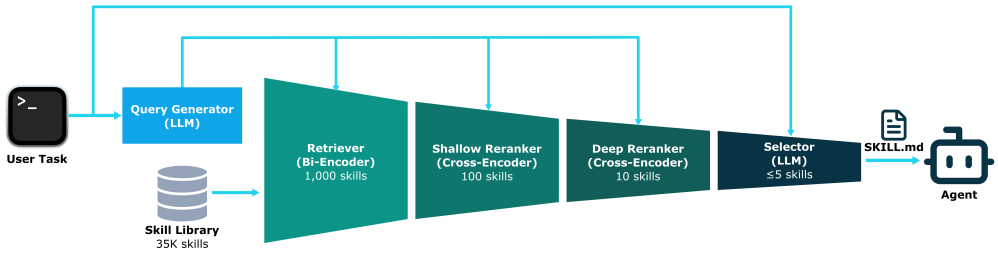

Figure 1: Overview of the SkillFlow pipeline. A user task is passed to an agent, which invokes SkillFlow to retrieve relevant skills from a library of 35K community-contributed skill definitions. The pipeline progressively narrows the candidate set through four stages: a bi-encoder retriever (1K skill candidates), a shallow cross-encoder reranker (100), a deep cross-encoder reranker (10), and an LLM-based selector ($\leq$5 skills). Retrieved skills are returned to the agent to augment task execution.

et al., 2026; Gao, 2026). Simply pre-loading all available skills is therefore impractical. What is needed is a mechanism that can selectively retrieve only the most relevant skills from a large and growing library.

In this work, we address this retrieval challenge and present *SkillFlow*, a multi-stage pipeline that, given a task description, efficiently identifies and retrieves relevant skills from a large library of community-contributed skill definitions. The pipeline consists of four stages: dense retrieval, shallow reranking, deep reranking, and final selection (Figure 1); each progressively refining the candidate set to balance recall and precision. We evaluate SkillFlow on two coding benchmarks: on SkillsBench, retrieved skills raise Pass@1 from 9.2% to 16.4% (+78.3%, $p_{\text{adj}} = 3.64 \times 10^{-2}$), reaching 84.1% of the oracle ceiling. On Terminal-Bench, however, agents readily adopt retrieved skills (70.1% use rate) yet show no performance gain, revealing that retrieval alone is insufficient when the corpus lacks high-quality skills for the target domain. This contrast suggests that the primary factor limiting skill-augmented agents is not retrieval but the quality and coverage of the underlying skill library.

Our contributions are as follows:

1. We frame agent skill acquisition as information retrieval over a large, real-world corpus of community-contributed SKILL.md definitions and, to our knowledge, are the first to study skill retrieval at this scale. We build a standard multi-stage cascade (dense retrieval $\rightarrow$ shallow reranking $\rightarrow$ deep reranking $\rightarrow$ selection) for this setting and claim no novel architecture, using a standard cascade by design, so the results reflect skill retrieval itself rather than bespoke choices.
2. We construct an openly available skill library by indexing ~35K community-contributed SKILL.md files from GitHub, providing a reusable resource for skill-augmented agents.
3. We evaluate SkillFlow on two benchmarks and, through a contrasting null result on Terminal-Bench, demonstrate that skill library quality, and not retrieval, is the primary bottleneck for skill-augmented agents. A structural comparison of 229 oracle skills against ~35K community-authored skills identifies the gap precisely: oracle skills contain significantly more executable code ($p_{\text{adj}} = 3.38 \times 10^{-12}$) and are roughly 2.5 times more likely to bundle runnable scripts ($p_{\text{adj}} = 5.57 \times 10^{-18}$), while length and documentation volume do not differ. These findings translate directly into measurable authoring guidelines about skill contribution and retrieval.

2 Related Work

Skill Acquisition and Tool Discovery. LLM agents increasingly leverage external capabilities at inference time. VOYAGER (Wang et al., 2023) synthesizes and retrieves skills via embedding similarity, but its library is self-generated rather than drawn from a shared corpus. Tool-calling approaches (Toolformer (Schick et al., 2023), Gorilla (Patil et al., 2023), ToolLLM (Qin et al., 2024), and Chen et al. (2025)) retrieve atomic API signatures, whereas SkillFlow retrieves higher-level procedural knowledge: structured SKILL.md bundles that may include multi-step instructions, executable scripts, and contextual references. Agent Skills (Anthropic, 2025) formalized this bundle format, and a rapidly growing ecosystem now hosts tens of thousands of community-contributed definitions. Despite this growth, existing skill-augmented agents assume a small, manually curated skill set or rely on brute-force enumeration; scalable retrieval over a large, heterogeneous skill library has not been addressed.

Multi-Stage Retrieval and Reranking. Retrieval-augmented generation (Lewis et al., 2020) has evolved from sparse methods like BM25 (Robertson & Zaragoza, 2009) to dense bi-encoders (Karpukhin et al., 2020), with cross-encoder rerankers trading throughput for accuracy (Nogueira & Cho, 2019; Nogueira et al., 2019). SkillFlow adopts this multi-stage architecture but operates over structured skill definitions (documents mixing natural-language descriptions with YAML metadata and embedded code) rather than homogeneous text passages, and uses a two-tier cross-encoder cascade to keep latency practical for interactive agent workflows.

3 SkillFlow

Given a task description t and a skill library $\mathcal{S}$ of N skills, the goal is to return a small subset $\mathcal{S}^* \subset \mathcal{S}$ of the most relevant skills under a latency budget compatible with interactive agent use. SkillFlow tackles this by progressively narrowing a large candidate set through four stages (dense retrieval, shallow reranking, deep reranking, and final selection), each refining the candidate skill set $\mathcal{C}_l$ (where l denotes the stage) to balance recall and precision. Each stage l retains the top k_l candidates from the previous stage's output, with $k_1 \gg k_2 \gg k_3 \geq |\mathcal{S}^*|$. Specific values are tuned via ablation experiments (Appendices A.4, A.5, and A.6).

Query Generation. For a given task t, we use an LLM to generate natural language queries that decompose the task into its constituent technologies, tools, and domains. Each query targets a different abstraction layer (core problem domain, programming language or framework, specific libraries, and supporting tools), so that the set collectively covers the skills an agent would need. Each pipeline stage l uses a query set $\mathcal{Q}^{(l)} = \{q_1^{(l)}, \ldots, q_{M_l}^{(l)}\}$ of M_l queries; when $M_l > 1$, per-query scores are aggregated by mean, max, or reciprocal rank fusion. Full prompt templates and examples are provided in the online appendix (Appendix A.2).

Dense Retrieval. Given a skill library $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ (Section 4.1), each skill description is precomputed into a d-dimensional dense embedding $\mathbf{e}_i = \mathrm{Enc}(s_i) \in \mathbb{R}^d$ using a bi-encoder, where d is determined by the encoder model. The first stage takes each query $q_j^{(1)} \in \mathcal{Q}^{(1)}$ and retrieves candidate skills by computing the cosine similarity between the query embedding $\mathbf{e}_{q_j} = \mathrm{Enc}(q_j^{(1)})$ and each skill embedding $\mathbf{e}_i$:

$$\mathrm{sim}(q_j^{(1)}, s_i) = \frac{\mathbf{e}_{q_j} \cdot \mathbf{e}_i}{\|\mathbf{e}_{q_j}\| \, \|\mathbf{e}_i\|}. \tag{1}$$

This stage prioritizes recall, ensuring that a broad set of potentially relevant skills is retrieved for further processing. The per-query result lists are then aggregated and deduplicated to form the first candidate set $\mathcal{C}_1$ of at most k_1 skills. The aggregation strategy (e.g., union of top results per query, score fusion) is configurable (Appendix A.4).

Shallow Reranking. We apply a lightweight cross-encoder model $\mathrm{CE}_{\mathrm{shallow}}$ to rerank the candidate skills in $\mathcal{C}_1$. For each query $q_j^{(2)} \in \mathcal{Q}^{(2)}$ and candidate skill, the model concatenates the query with the skill content truncated to L_{shallow} characters:

$$r_{\mathrm{shallow}}(q_j^{(2)}, s_i) = \mathrm{CE}_{\mathrm{shallow}}\big([q_j^{(2)}; \mathrm{trunc}(s_i, L_{\mathrm{shallow}})]\big). \tag{2}$$

The top-k_2 candidates are retained:

$$\mathcal{C}_2 = \text{top-k}_2\big(\mathcal{C}_1, r_{\mathrm{shallow}}\big). \tag{3}$$

Deep Reranking. Scoring all $|\mathcal{C}_1|$ candidates against full skill content is prohibitively expensive. The shallow reranker first reduces the candidate set, making full-content scoring tractable on the smaller set $\mathcal{C}_2$. The deep reranking stage employs a heavier cross-encoder $\mathrm{CE}_{\mathrm{deep}}$ that attends to skill content truncated at a much larger limit $L_{\mathrm{deep}} \gg L_{\mathrm{shallow}}$, enabling a more thorough evaluation of relevance:

$$r_{\mathrm{deep}}(q_j^{(3)}, s_i) = \mathrm{CE}_{\mathrm{deep}}\big([q_j^{(3)}; \mathrm{trunc}(s_i, L_{\mathrm{deep}})]\big). \tag{4}$$

The top-k_3 candidates are retained:

$$\mathcal{C}_3 = \text{top-k}_3\big(\mathcal{C}_2, r_{\mathrm{deep}}\big). \tag{5}$$

Final Selection. The final stage uses an LLM-based filter f_{select} that evaluates each candidate in $\mathcal{C}_3$ for *relevancy*, whether the skill is topically relevant to the task t, and retains only those that pass:

$$\mathcal{S}^* = f_{\mathrm{select}}(t, \mathcal{C}_3), \quad |\mathcal{S}^*| \leq |\mathcal{C}_3|. \tag{6}$$

Unlike the previous stages, which output a fixed number of candidates, the selector outputs a variable number of skills $|\mathcal{S}^*|$, compacting the injected set for the downstream agent. Optionally, a second *specificity filter* can be chained to further assess whether surviving skills provide actionable, domain-specific guidance beyond what the agent can derive from its training data (Appendix A.6).

4 Experiments and Results

4.1 Datasets

Skill Library. We construct a skill library by collecting skills from SkillsMP, an open marketplace of community-contributed agent skills. Skills are enumerated via the SkillsMP API. For each skill, the associated GitHub repository is downloaded and only those containing a valid SKILL.md file are retained. Of these, 35,129 contain a top-level SKILL.md and form the final indexed library; the remainder download successfully but ship no SKILL.md and are discarded. Each skill entry includes a name, a natural language description, and associated metadata. The full skill descriptions are embedded using an encoder-based embedding model to support dense retrieval. Details of the crawling pipeline and corpus statistics are provided in the online appendix (Appendix A.1).

SkillsBench. SkillsBench (Li et al., 2026) is a benchmark of 87 tasks, each paired with one or more human-authored *oracle skills*, curated skill definitions known to be relevant to the task (229 total). Skills are co-authored alongside tasks by domain experts and validated through automated checks and human review, establishing topical relevance (the skills address the right domain and techniques), though not guaranteed benefit on every task-agent combination. We treat these pairings as retrieval ground truth by injecting the oracle skills into the Skill Library and measuring whether SkillFlow can recover them (Experiments 1, 2, and 3).

Terminal-Bench. Terminal-Bench (Merrill et al., 2026) is a benchmark of terminal-based coding tasks. It consists of 89 tasks that test an agent's ability to perform operations in a terminal environment. Unlike SkillsBench, Terminal-Bench does not provide oracle skill assignments, so we use it for end-to-end evaluation only (Experiments 1 and 2).

4.2 Settings

For query generation (Appendix A.2), we use OpenAI gpt-4o-mini (Hurst et al., 2024). The dense retriever generates M_1=5 queries per task and uses Hugging Face BAAI/bge-base-en-v1.5 (Xiao et al., 2023) as the bi-encoder, retrieving the top $k_1 = 1000$ candidates in total. The shallow reranker uses SBERT cross-encoder/ms-marco-MiniLM-L-6-v2 (Reimers & Gurevych, 2019) as a cross-encoder with $L_{\text{shallow}} = 512$ characters and a single query (M_2=1), retaining the top $k_2 = 100$ candidates. The deep reranker uses Hugging Face BAAI/bge-reranker-v2-m3 (Chen et al., 2024) as a cross-encoder with $L_{\text{deep}} = 4096$ characters, also with M_3=1, retaining the top $k_3 = 10$ candidates. The final selector uses OpenAI gpt-4o-mini (Hurst et al., 2024) to select up to 5 skills. For end-to-end benchmark evaluation, we use Codex CLI with OpenAI gpt-5-mini (Singh et al., 2025) at medium reasoning effort as the primary agent, and Claude Code with Claude Haiku 4.5 as a second agent on SkillsBench to test whether the gains transfer across harnesses and vendors (Section 4.3). We report results using the best-performing pipeline configuration. Additional experiments on key hyperparameters (e.g., k values, query count M, and model choices) are provided in the online appendix. For the benchmark harness, we rely on Harbor (Team, 2026) with the Daytona Sandboxes cloud service. Except proprietary OpenAI models, all language model components in SkillFlow are run on an Nvidia RTX A5000 GPU.

Metrics. Each condition is run 3 times. For end-to-end evaluation (Experiments 1–2), we report Pass@k (the probability that at least one of k attempts succeeds, estimated from the 3 runs via the unbiased combinatorial estimator (Chen et al., 2021)) and skill use rate (the fraction of tasks where the agent uses at least one retrieved skill). For retrieval evaluation (Experiment 3), we report recall@k (fraction of oracle skills in the top k), precision@k (fraction of top k that are oracle skills), mean reciprocal rank (MRR, the reciprocal of the rank of the first oracle skill), and hit@k (fraction of tasks with at least one oracle skill in the top k), all averaged across tasks. All confidence intervals are 95% bootstrap (10,000 resamples over tasks). Pairwise significance is assessed via paired bootstrap tests. Reported p_{adj} values are Holm-Bonferroni corrected for multiple comparisons within each benchmark. Full statistical details are in the online appendix (Appendix A.3).

4.3 Experiment 1: Downstream Benchmark Performance

To evaluate end-to-end effectiveness, we measure whether skills retrieved by SkillFlow improve agent task completion. We benchmark on 65 SkillsBench tasks and 88 Terminal-Bench tasks after removing problematic tasks (Appendix A.8). Both benchmarks are evaluated under four conditions: (1) no skills, (2) Vercel skills (a third-party skill retrieval API; Appendix A.7), (3) SkillFlow-retrieved skills, and (4) oracle skills (SkillsBench only, as Terminal-Bench does not provide oracle assignments).

Downstream gains depend on skill library coverage. Table 1 summarizes the results. On SkillsBench, where oracle skills exist in the library, SkillFlow raises Pass@1 from 9.2% to 16.4% (+78.3%), a statistically significant improvement ($p_{\text{adj}} = 3.64 \times 10^{-2}$; Cohen's $h = 0.22$, small effect), reaching 84.1% of the oracle ceiling (19.5%); the trend holds at Pass@3 (29.2% vs. 16.9% baseline), while Vercel shows no significant improvement (9.7%, $p_{\text{adj}} \geq 0.05$). Steps and cost per task remain comparable across conditions, indicating that gains stem from skill quality rather than increased computation. The gain replicates on a second agent: on Claude Code with Claude Haiku 4.5 (Table 1, three runs), the Oracle > SkillFlow > no-skill ordering holds with a +159% relative gain (Pass@1 6.7% → 17.4%, $p_{\text{adj}} = 2.00 \times 10^{-4}$, Cohen's $h = 0.34$; oracle ceiling 30.3%). The selector affects only context size: injecting the deep reranker's top-10 skills directly matches full SkillFlow at equal cost (Pass@1 15.9% vs. 16.4%, paired bootstrap p=0.91; Table 18).

On Terminal-Bench, however, the no-skill baseline is already strong (Pass@1 = 34.8%, Pass@3 = 46.6%), and no condition produces a significant difference: neither SkillFlow (31.8%, $p_{\text{adj}} \geq 0.05$) nor Vercel (32.6%, $p_{\text{adj}} \geq 0.05$) improve over it. This contrast (gains where oracle skills exist, none where the library lacks coverage, and holding across two

Table 1: Benchmark performance comparison. Pass@k: probability of at least one success in k independent attempts; Pass^k: probability that all k attempts succeed. Oracle skills are the curated task-skill pairs provided by SkillsBench. $^{*}p_{\text{adj}} < 0.05$, $^{**}p_{\text{adj}} < 0.01$ vs. no skills baseline.

Model	Condition	Pass@1	Pass@3	Pass^3	Steps/Task	Cost/Task
SkillsBench						
GPT-5-mini	*Oracle*	*19.5** [12.3, 27.2]	*35.4* [24.6, 47.7]	*6.2* [1.5, 12.3]	*20.1*	*$0.028*
	No Skills	9.2 [4.1, 14.9]	16.9 [7.7, 26.2]	0.0 [0.0, 0.0]	**18.7**	$0.026
	Vercel	9.7 [4.6, 15.4]	20.0 [10.8, 30.8]	1.5 [0.0, 4.6]	20.9	$0.029
	SkillFlow (Ours)	**16.4*** [9.7, 23.6]	**29.2** [18.5, 40.0]	**4.6** [0.0, 10.8]	18.8	**$0.025**
Claude Haiku 4.5	*Oracle*	*30.3*** [21.5, 39.5]	*44.6* [32.3, 56.9]	*13.8* [6.2, 23.1]	*55.4*	*$0.428*
	No Skills	6.7 [2.6, 11.8]	12.3 [4.6, 21.5]	0.0 [0.0, 0.0]	**50.8**	$0.418
	SkillFlow (Ours)	**17.4**** [10.3, 25.1]	**29.2** [18.5, 40.0]	**4.6** [0.0, 10.8]	51.4	**$0.404**
Terminal-Bench						
GPT-5-mini	No Skills	34.8 [26.1, 43.6]	46.6 [36.4, 56.8]	20.5 [12.5, 29.5]	**21.1**	**$0.030**
	Vercel	32.6 [24.2, 41.3]	45.5 [35.2, 55.7]	**21.6** [13.6, 30.7]	21.2	$0.031
	SkillFlow (Ours)	31.8 [23.9, 40.2]	46.6 [36.4, 56.8]	17.0 [9.1, 25.0]	23.0	$0.034
	SkillFlow-specific (Ours)	**34.9** [26.7, 43.2]	**48.0** [37.8, 58.0]	21.0 [13.4, 29.0]	24.2	$0.035

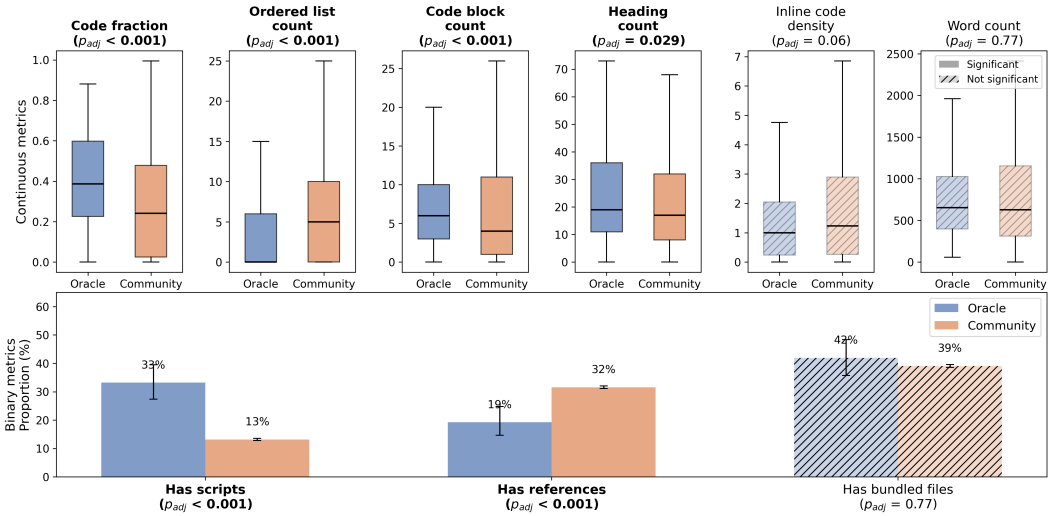

Figure 2: Structural proxy metrics: oracle (n=229) vs. community (n=35,129) skills. Subplots sorted by significance. p-values from two-sided Mann-Whitney U tests, adjusted via Holm-Bonferroni. Hatched boxes/bars indicate non-significant differences ($p_{\text{adj}} \geq 0.05$). Outliers suppressed for clarity.

agents) suggests that skill quality, not retrieval, is the primary bottleneck. Table 1 also reports a SkillFlow-specific variant with additional specificity filtering. We analyze this condition in Section 5.

Characterizing the quality gap. We compare structural properties of 229 oracle skills against the 35,129 indexed community skills using automated proxy metrics (Figure 2; two-sided Mann-Whitney U, Holm-Bonferroni adjusted). Oracle skills are not significantly longer ($p_{\text{adj}} \geq 0.05$), but devote a larger fraction to fenced code blocks (0.39 vs. 0.24, $p_{\text{adj}} = 3.38 \times 10^{-12}$) and are roughly 2.5 times more likely to bundle executable scripts (33.2% vs. 13.2%, $p_{\text{adj}} = 5.57 \times 10^{-18}$). Conversely, community skills favor ordered lists and reference documents, implying a more procedural, text-heavy style. What distinguishes effective skills is not length or documentation volume, but the density of runnable code and bundled artifacts.

Table 2: Skill retrieval and agent use statistics. Tasks Retrieved: percentage of tasks for which ≥1 skill was retrieved. Oracle Skills Retrieved: percentage of oracle skills retrieved from the library. Tasks Used: percentage of tasks where the agent used ≥1 retrieved skill. Vercel maintains its own proprietary skill index that does not include SkillsBench oracle skills. Terminal-Bench does not provide oracle skill assignments. $^{*}p_{\text{adj}} < 0.05$; $^{**}p_{\text{adj}} < 0.01$ on Tasks Used vs. Vercel baseline.

Condition	Mean Skills Retrieved/Task	Tasks Retrieved (%)	Oracle Skills Retrieved (%)	Tasks Used (%)
SkillsBench				
Oracle	*2.5*	*100.0* [100.0, 100.0]	*100.0* [100.0, 100.0]	*69.2* [62.6, 75.9]
Vercel	0.9	87.5 [82.8, 92.2]	—	40.6 [33.9, 47.9]
SkillFlow top-1 (Ours)	0.9	91.7 [87.5, 95.3]	37.6 [32.4, 43.0]	**68.8**** [62.5, 75.0]
SkillFlow (Ours)	2.8	**92.2** [88.0, 95.8]	**61.8** [56.5, 66.9]	**68.8**** [62.0, 75.0]
Terminal-Bench				
Vercel	0.9	**90.9** [87.1, 94.3]	—	22.3 [17.4, 27.7]
SkillFlow top-1 (Ours)	0.9	89.8 [86.0, 93.2]	—	**72.7**** [67.4, 78.0]
SkillFlow (Ours)	1.5	87.5 [83.3, 91.3]	—	70.1** [64.4, 75.4]

Table 3: Stage-level retrieval performance. Recall (R), precision (P), and mean reciprocal rank (MRR) across each stage of the SkillFlow retrieval pipeline. Inapplicable cells (where k exceeds the stage output size) are marked with dashes. K output denotes the number of candidates produced by each stage.

Stage	K output	Metric	@1	@5	@10	@50	@100	@500	@1000	MRR
Retriever	1000	R	0.174 [0.12, 0.24]	0.376 [0.30, 0.46]	0.469 [0.39, 0.55]	0.603 [0.52, 0.69]	0.670 [0.59, 0.75]	0.859 [0.80, 0.91]	0.905 [0.85, 0.95]	0.487 [0.40, 0.58]
		P	0.379 [0.28, 0.48]	0.182 [0.14, 0.22]	0.122 [0.10, 0.15]	0.032 [0.03, 0.04]	0.018 [0.02, 0.02]	0.005 [0.00, 0.01]	0.002 [0.00, 0.00]	
Shallow Reranker	100	R	0.259 [0.19, 0.33]	0.460 [0.38, 0.55]	0.520 [0.43, 0.61]	0.687 [0.61, 0.76]	0.776 [0.71, 0.84]	—	—	0.587 [0.50, 0.68]
		P	0.494 [0.39, 0.60]	0.216 [0.18, 0.26]	0.130 [0.11, 0.16]	0.034 [0.03, 0.04]	0.019 [0.02, 0.02]	—	—	
Deep Reranker	10	R	0.271 [0.20, 0.34]	0.499 [0.42, 0.58]	0.595 [0.51, 0.68]	—	—	—	—	0.634 [0.54, 0.72]
		P	0.540 [0.44, 0.64]	0.237 [0.20, 0.28]	0.147 [0.12, 0.17]	—	—	—	—	
Selector	≤5	R	0.273 [0.20, 0.34]	0.455 [0.38, 0.53]	—	—	—	—	—	0.639 [0.55, 0.73]
		P	0.563 [0.46, 0.67]	0.221 [0.18, 0.26]	—	—	—	—	—	

4.4 Experiment 2: Skill Retrieval and Usage

To understand whether agents actually use retrieved skills, we compare SkillFlow against Vercel, a third-party skill retrieval API (Appendix A.7), measuring retrieval and skill use statistics across both benchmarks. For each task, each method retrieves skills from the library. The agent then decides which (if any) to load into its context. On SkillsBench, oracle task–skill pairs serve as relevancy ground truth. Terminal-Bench lacks such annotations.

SkillFlow achieves higher retrieval and use rates. Table 2 reports the results. On SkillsBench, SkillFlow retrieves 61.8% of oracle skills from the 35K library. This comparison is unavailable for Vercel, whose proprietary index does not include the SkillsBench oracle skills. Agents use SkillFlow skills at a 68.8% rate ($p_{\text{adj}} < 1 \times 10^{-6}$ vs. Vercel) with an average of 2.8 skills retrieved per task, approaching the oracle use rate of 69.2%, indicating that the pipeline consistently surfaces skills the agent judges useful. On Terminal-Bench, SkillFlow retrieves skills for 87.5% of tasks compared to Vercel's 90.9%, and agents use SkillFlow skills at a significantly higher rate (70.1% vs. 22.3%, $p_{\text{adj}} < 1 \times 10^{-6}$).

Even when restricted to a single top-ranked skill (matching Vercel's cardinality), SkillFlow achieves significantly higher use rates (68.8% vs. 40.6% on SkillsBench; 72.7% vs. 22.3% on Terminal-Bench; both $p_{\text{adj}} < 1 \times 10^{-6}$), confirming that the advantage stems from retrieval quality.

4.5 Experiment 3: Stage-Level Retrieval Performance

This experiment evaluates SkillFlow's retrieval quality at each pipeline stage. We use the 229 oracle skills from the 87 SkillsBench tasks as retrieval ground truth and measure their overlap with SkillFlow's output at each stage.

Table 4: Stage-ablation retrieval metrics on SkillsBench (87 tasks). MRR, R@5, R@10, P@10, and Hit@10 (fraction of tasks with ≥1 oracle skill in top 10) reported. The first three rows compare retrieval methods. Subsequent rows add pipeline stages cumulatively. The selector emits at most 5 skills, so its @10 metrics are inapplicable (dashes); it raises MRR while lowering R@5.

Pipeline	MRR	R@5	R@10	P@10	Hit@10
First-stage alternatives					
BM25 only	0.266 [0.19, 0.35]	0.211 [0.14, 0.29]	0.238 [0.17, 0.32]	0.055 [0.04, 0.07]	0.391 [0.29, 0.49]
Hybrid (Dense + BM25)	0.522 [0.43, 0.61]	0.411 [0.33, 0.49]	0.480 [0.40, 0.56]	0.114 [0.09, 0.14]	0.713 [0.61, 0.81]
Dense only	0.553 [0.46, 0.64]	0.415 [0.34, 0.50]	0.477 [0.39, 0.56]	0.121 [0.10, 0.15]	0.713 [0.61, 0.81]
Cumulative pipeline					
+ Shallow reranker (1–2)	0.587 [0.50, 0.68]	0.460 [0.38, 0.55]	0.520 [0.43, 0.61]	0.130 [0.11, 0.16]	0.724 [0.63, 0.82]
+ Deep reranker (1–3)	0.634 [0.54, 0.72]	**0.499** [0.42, 0.58]	**0.595** [0.51, 0.68]	**0.147** [0.12, 0.17]	**0.793** [0.70, 0.87]
+ Selector (1–4)	**0.639** [0.55, 0.73]	0.455 [0.38, 0.53]	—	—	—

Reranking improves top-heavy ranking, while selection trades recall for precision. Table 3 reports recall, precision, and MRR at each stage. The dense retriever achieves R@1000 of 0.905. The shallow reranker compresses to 100 candidates while boosting R@1 by 48.9% and MRR from 0.487 to 0.587. The deep reranker narrows to 10 candidates with MRR of 0.634. The selector then trades recall for precision, lowering R@5 but achieving the highest P@1 (0.563), consistent with a precision-oriented compaction gate; this recall reduction does not lower downstream Pass@1, since the no-selector ablation matches full SkillFlow (Section 4.3). A stage ablation (Table 4) further confirms that progressive refinement outperforms alternative first-stage methods (BM25, hybrid), with the deep reranker achieving the best R@5 (0.499) and Hit@10 (0.793); adding the selector raises MRR to 0.639 while lowering R@5 to 0.455, the same precision-for-recall trade shown in Table 3.

Performance Impact of Query Generation. Figure 3 shows the effect of the number of generated queries (M) on each pipeline stage. Multi-query generation (M=5) improves retriever pass-through recall by +3.0% by casting a wider net, but degrades reranker performance (−10.7% at the deep reranker, $p_{\text{adj}} = 4.92 \times 10^{-2}$ for M=3). M=1 achieves the highest MRR at every stage (panel b), confirming that on small, already-filtered candidate sets, additional queries introduce noise. The R@k curves (panels c–e) show that M=5 overtakes M=1 only beyond k≈224 at the retriever, while M=1 dominates at all reranker cutoffs. These results motivate our design: M=5 for retrieval to maximize recall, M=1 for reranking to maximize ranking quality.

Structure-aware indexing raises the recall ceiling. Motivated by description-only embedding beating full-content embedding (Table 9), we test a structure-aware index that encodes the YAML, prose, and code sections of each SKILL.md as three independent dense indices fused at the score level. On a matched single-query eval, the 3-way split raises recall at every cutoff (R@1000 from 0.843 to 0.925, R@100 by up to +13.2pp) at a small cost in top-1 ranking, while a code-specialized encoder does not help in this natural-language-driven setting (Table 12). Because this gain sits at the first-stage recall ceiling that bounds every downstream stage, the sectioned index is a promising representation; we leave its end-to-end evaluation to future work.

4.6 Qualitative Analysis: Why Gains Are Bounded

Three analyses explain why gains stay bounded even where SkillFlow retrieves well (full tables in the appendix). *Retrieval blind spots:* An oracle skill's value often lies in its bundled runnable scripts, not its prose: in the court-form task (Table 21) the oracle skill helps only because it ships executable scripts that a description-based retriever and a truncated reranker never see: the structural gap of Figure 2. *Query-generation gap:* Some oracle skills are generic utilities the task never names; the out-of-query share rises from 4% among top-5 oracles to 48% among the worst-ranked (Table 13), e.g. a Civilization 6 task needing

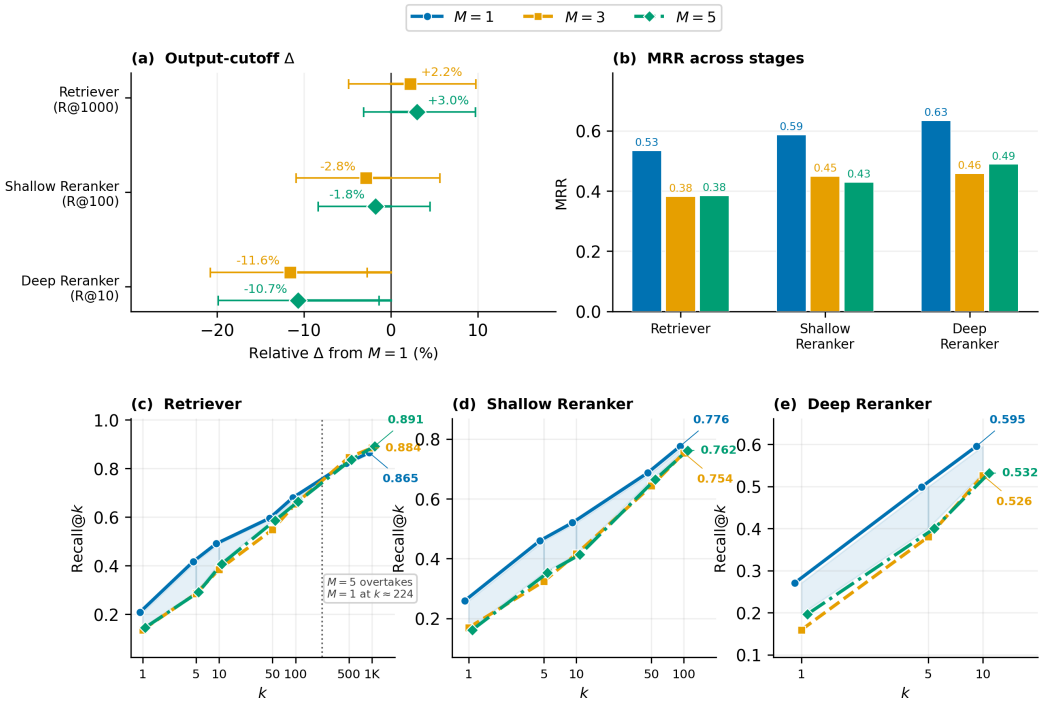

Figure 3: Multi-query impact across pipeline stages. (a) Relative change in pass-through recall (R@k_{output}) from the M=1 baseline. (b) MRR by stage, showing that M=1 yields the best top-heavy ranking at every stage. (c–e) Recall@k curves for each stage; the dashed line in (c) marks where M=5 overtakes M=1. Multi-query improves retriever coverage but degrades reranker discrimination.

the generic `hex-grid-spatial` skill, motivating artifact-aware query generation (Future directions, Section 5). *Help vs. hurt:* Across the SkillsBench runs, retrieved skills improve 16 of 64 tasks and regress 4 (paired bootstrap p=0.02; Table 23); agents open non-oracle skills only 17% of the time versus 42% for oracle skills, and gains concentrate where an oracle skill was recovered, motivating skill adaptation as future work.

5 Discussion

SkillFlow enables scalable retrieval. The multi-stage pipeline efficiently trades breadth for precision: the dense retriever captures 90.5% of oracle skills across 35K candidates, and each subsequent stage improves top-heavy precision while maintaining comparable steps and cost per task. The full pipeline completes in ∼35 seconds per task (median), dominated by the deep reranker (∼26s); per-stage latency is reported in Table 5. The single-query configuration (M=1 for reranking) offers the best latency–quality balance, keeping end-to-end time well within typical agent task durations and making large skill libraries practical for real-time use. This latency is largely hardware-dependent (the deep reranker runs locally on an RTX A5000) and reducible with a smaller cross-encoder or a hosted reranker API.

Retrieval gains are bounded by what the library contains. On SkillsBench, where oracle skills exist, SkillFlow raises Pass@1 by 78.3% ($p_{\text{adj}} = 3.64 \times 10^{-2}$), reaching 84.1% of the oracle ceiling. On Terminal-Bench, where the library lacks domain coverage, no condition differs significantly from baseline ($p_{\text{adj}} \geq 0.05$, Cohen's $h < 0.07$), confirming that the bottleneck is what is *in* the library, not how we search it. The structural quality analysis (Figure 2) reinforces this: effective skills are distinguished by the density of runnable code and bundled artifacts, a gap that retrieval alone cannot close. Retrieval itself is not

Table 5: Per-stage wall-clock latency (seconds) on 87 SkillsBench tasks, Nvidia RTX A5000 GPU.

Stage	Median	Mean	P95
Stage 1: Dense Retriever	3.3	3.4	4.0
Stage 2: Cross-Encoder Reranker	1.6	1.6	2.2
Stage 3: Deep Reranker	25.6	26.4	33.1
Stage 4: LLM Selector	2.7	3.6	8.5
Total	**35.0**	**35.0**	**43.8**

the limiting factor: SkillFlow recovers 90.5% of oracle skills at the retriever (Section 4.5) and reaches 84.1% of the oracle Pass@1 ceiling (Section 4.3), so the low absolute ceiling reflects downstream factors (agent non-adoption, task difficulty, and variance) and is model-dependent (19.5% on GPT-5-mini, 30.3% on Claude Haiku 4.5), so we expect it to rise with stronger agents, a larger library, and the retrieval upgrades above. Since Terminal-Bench has no oracle assignments, deriving validated oracle skills for such domains (SkillsBench used 105 contributors) is an open community direction, not a limitation of retrieval.

Filtering low-quality skills. We augment SkillFlow with a specificity gate, sequential relevancy and specificity LLM filters, producing the SkillFlow-specific variant (Table 1). The effect is negligible on Terminal-Bench ($p_{\text{adj}} \geq 0.05$), and the reason is placement: by the specificity stage the relevancy pass has already removed the topically irrelevant candidates, leaving little to change, so the specificity filter is a cheap safety check rather than a load-bearing stage. The remaining quality gap (code density, bundled artifacts) is structural rather than semantic, and thus not easily captured by an LLM-based filter.

Future directions. Natural extensions include expanding the library through automated extraction from repositories or agent interaction logs, peer-to-peer skill sharing, and *interleaved retrieval*, retrieving skills dynamically during task execution rather than only beforehand. On the retrieval side, *artifact-aware query generation*, inspecting each task's files to enumerate implied tools and issuing a query per dependency, would reach the out-of-query oracle skills that task descriptions miss (Section 4.6), and per-candidate *snippet summarization* could preserve signal in a smaller reranker window; code-heavy benchmarks such as FreshStack are a natural evaluation target. Because retrieved skills are only sometimes useful as-is, *skill adaptation* (editing retrieved skills to the task) and automating high-quality skill and oracle derivation are promising ways to raise the ceiling and extend the method to sparse domains.

Limitations. We evaluate two agent models (Codex CLI with GPT-5-mini and Claude Code with Claude Haiku 4.5) across two vendors and two benchmarks; broader coverage of architectures and domains remains future work. Our corpus audit finds only about 32% of community skills are runnable-complete (Section A.1), motivating stronger vetting before skills enter the corpus; we do not verify runtime correctness, and the quality-gap analysis relies on structural proxies and an in-context LLM judge rather than human judgments. The skill library was constructed at a fixed point in time; a more up-to-date library would likely improve coverage. Retrieval also sees each skill only through its description and a truncated window, so bundled scripts and long-tail dependencies stay invisible; snippet summarization and artifact-aware query generation are the natural remedies, and we leave both to future work.

Ethics Statement

Aggregating community-contributed skills at scale raises attribution and licensing concerns. Skills in the library are sourced from public GitHub repositories under various open-source licenses. Our crawling pipeline preserves repository metadata but does not currently enforce license-specific constraints on redistribution or modification. Practitioners deploying

SkillFlow should implement license filtering appropriate to their use case. Additionally, a centralized skill library could be a vector for skill poisoning, malicious skills crafted to induce harmful agent behavior, motivating future work on skill provenance verification and content safety filtering.

Reproducibility Statement

To support reproduction, we release the crawler and skill metadata/index, the retrieval code (dense retrieval, shallow and deep rerankers, and the LLM selector), the query-generation prompts and cached generated queries, the benchmark harness scripts, the task-exclusion list, and the statistical analysis scripts, together with instructions to reconstruct the ~35K-skill corpus from its source GitHub URLs. All artifacts are available at https://github.com/IBPA/skill-flow. Because upstream repositories may change or be removed, we record per-skill content hashes and download timestamps so that a reconstructed corpus can be validated against the version used in our experiments.

Disclosure of LLM usage. LLMs were used for code implementation, with all code reviewed by the authors. LLMs were also used for refining paper writing with human proofreading. Experimental results, tables, figures, and references were generated by the authors manually or programmatically.

Acknowledgments

This work was supported by the USDA-NIFA AI Institute for Next Generation Food Systems (AIFS), USDA-NIFA award number 2020-67021-32855.

References

Anthropic. Agent skills overview, 2025. URL https://agentskills.io/home.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Annual Meeting of the Association for Computational Linguistics*, pp. 2318–2335. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.137.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv.org*, 2021. doi: 10.48550/arXiv.2107.03374.

Yi-Chang Chen, Po-Chun Hsu, Chan-Jan Hsu, and Da-shan Shiu. Enhancing function-calling capabilities in LLMs: Strategies for prompt formats, data integration, and multilingual translation. In Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 99–111, Albuquerque, New Mexico, apr 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-industry.9. URL https://aclanthology.org/2025.naacl-industry.9/.

Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, D. Terzopoulos, Yejin Choi, et al. Agent AI: Surveying the horizons of multimodal interaction. *arXiv.org*, 2024. doi: 10.48550/arXiv.2401.03568. URL http://arxiv.org/abs/2401.03568.

Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Jaime G. Carbonell, Jörg Siekmann, Jörg P. Müller, Michael J. Wooldridge, and Nicholas R. Jennings (eds.), *ATAL*, pp. 21–35. Springer Berlin Heidelberg, 1996. doi: 10.1007/BFb0013570. URL http://link.springer.com/10.1007/BFb0013570.

Jude Gao. AGENTS.md outperforms skills in our agent evals, 2026. URL https://vercel.com/blog/agents-md-outperforms-skills-in-our-agent-evals.

OpenAI Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

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

Patrick Lewis, Ethan Perez, Aleksandara Piktus, F. Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, M. Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Neural Information Processing Systems*, pp. 9459–9474, 2020.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, B. You, Haotian Shen, Jiankai Sun, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks, 2026. URL http://arxiv.org/abs/2602.12670.

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, I. Bercovich, Lin Shi, J. Shin, Thomas Walshe, E. K. Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *arXiv.org*, 2026. doi: 10.48550/arXiv.2601.11868. URL https://openreview.net/forum?id=a7Qa4CcHak.

Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv.org*, 2019. URL https://arxiv.org/abs/1901.04085.

Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-stage document ranking with bert. *arXiv.org*, 2019. URL https://arxiv.org/abs/1910.14424.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Neural Information Processing Systems*, volume 37, pp. 126544–126565. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.52202/079017-4020.

Derek J. de Solla Price. Automata and the origins of mechanism and mechanistic philosophy. *Technology and Culture*, 5(1):9, 1964. doi: 10.2307/3101119. URL https://www.jstor.org/stable/3101119?origin=crossref.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, X. Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2307.16789. URL https://openreview.net/forum?id=dHng2O0Jjr.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*, pp. 3980–3990. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1410.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 2009. doi: 10.1561/1500000019.

Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, third edition edition, 2020.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, R. Raileanu, M. Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36*, volume 36, pp. 68539–68551. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.52202/075280-2997.

Aaditya K. Singh, A. Fry, Adam Perelman, A. Tart, A. Ganesh, Ahmed El-Kishky, A. McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. OpenAI GPT-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, jan 2026. URL https://github.com/harbor-framework/harbor.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2023. doi: 10.48550/arXiv.2305.16291. URL http://arxiv.org/abs/2305.16291.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv.org*, 2022.

Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *Knowledge engineering review (Print)*, 10(2):115–152, 1995. doi: 10.1017/S0269888900008122. URL https://www.cambridge.org/core/product/identifier/S0269888900008122/type/journal_article.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 641–649. ACM, 2023. doi: 10.1145/3626772.3657878.

A Appendix

A.1 Skill Library Construction

Skills were collected from SkillsMP, an open marketplace hosting over 80,259[1] agent skills. Each skill is a structured bundle containing a Markdown document (SKILL.md) with YAML frontmatter containing metadata (name, description, allowed tools) and free-form instructional content such as code examples, reference material, and tutorials.

Crawling Pipeline. The collection pipeline operated in three stages. First, skills were enumerated via the SkillsMP API with page-based pagination. For each skill, metadata was recorded including name, description, author, tags, star count, and associated GitHub repository URL. Second, the corresponding GitHub repository (or subdirectory) for each skill was downloaded as a ZIP archive via the GitHub API. A configurable repository size limit of 50 MB was enforced to exclude oversized repositories. Downloads used rate-limited asynchronous HTTP requests (1.0 s inter-request delay) with exponential backoff retry (1–10 s, up to 3 attempts) for transient failures. The pipeline supported resumable crawling via persisted sync state. Third, only skills containing a valid SKILL.md file were retained. Skills hosted on deleted or inaccessible repositories (HTTP 404/403) were permanently skipped.

Corpus Statistics. Table 6 summarizes the collection results. Of the 43,660 skills processed, 7,703 were excluded due to the 50 MB repository size filter and 91 failed due to deleted or inaccessible repositories, yielding 35,866 downloaded skills, of which 35,129 carry a valid top-level SKILL.md and are embedded into the retrieval index.

Runnable-completeness audit. The structural analysis (Figure 2) shows a code-density gap; here we quantify how much of the corpus is actually functional. Structurally, 78.1% of the 35,129 indexed community skills are code-bearing (containing at least a fenced code block or a bundled script), but only 13.2% bundle an executable script (vs. 33.2% of oracle skills). To go beyond structure, an LLM judge scored 300 stratified-sampled skills in context

[1] As of 1/24/2026 in https://skillsmp.com

Table 6: Skill corpus collection statistics.

Metric	Count
Skills processed	43,660
Excluded (repo > 50 MB)	7,703
Failed (deleted/inaccessible)	91
Downloaded	35,866
Indexed (valid SKILL.md)	35,129

Table 7: Runnable-completeness audit of 300 stratified-sampled community skills, scored by an in-context LLM judge. Runnable-complete: ships complete runnable code with no missing bundled files. 95% Wilson confidence interval.

Metric	Value
Code-bearing (corpus, n=35,129)	78.1%
Bundles executable scripts	13.2%
Has fenced code block	76.5%
Has any bundled file	39.1%
Runnable-complete (sample, n=300)	32.3% [27.3%, 37.8%]
judge tier: functional	98
judge tier: partial	26
judge tier: reference-only	176

(SKILL.md plus the bundled-file listing) for code-soundness, missing files, and purpose-alignment (Table 7). By this audit, 32.3% are *runnable-complete*, shipping complete runnable code with no missing bundled files (95% CI $[27.3, 37.8]$), while 59% are reference-only prose and 9% intend to ship code that is incomplete or references absent files. At scale this is roughly 11.5K runnable-complete skills, still a substantial resource, and it quantifies the corpus-quality gap that motivates our work. Verifying that this code produces exactly correct output would require executing every skill and is out of scope.

Data Format. Each skill is stored as a directory containing a SKILL.md file in structured Markdown format with YAML frontmatter (name, description, allowed tools) followed by instructional content. Skill metadata (including name, source URL, GitHub URL, author, tags, star count, content hash, and download timestamp) is persisted in a JSON index for downstream retrieval system indexing.

A.2 Query Generation Details

Given a task description, query generation produces concise natural language queries that capture the core technical skills required. These queries serve as inputs to the downstream retrieval and reranking stages. Two prompt variants are used depending on the pipeline stage: a **single-query prompt** (Figure 4), used by the reranker and deep reranker stages ($M = 1$), which produces one concise query capturing the primary technology or technique; and a **multi-query prompt** (Figure 5), used by the retriever stage ($M > 1$), which produces M diverse queries covering different layers of the task (core domain, language/framework, specific libraries, and supporting tools). Both prompts use gpt-4o-mini (Hurst et al., 2024) with a maximum output length of 200 tokens and temperature 0.0. Table 8 shows example outputs for both variants on a single task.

Single-Query Prompt

```
You are a search query generator. Given a detailed task instruction for a coding
agent, generate a concise search query (1-2 sentences, under 200 characters) that
captures the core technical skill needed. Focus on the primary technology, tool,
or technique required. Omit file paths, specific data values, and implementation
details.
```

Figure 4: Single-query prompt used by the reranker and deep reranker stages ($M = 1$).

Multi-Query Prompt

```
You are a search query generator for a skill retrieval system. A skill is a
self-contained reference document that teaches an AI agent how to use a specific
technology, tool, library, or technique.
Given a task instruction, generate {num_queries} diverse search queries that would
each match a different skill an agent might need. Each query should be 1-2
sentences, under 200 characters.
Guidelines for generating queries:
1. Cover different LAYERS of the task: the core problem domain, the programming
language/framework, specific libraries or file formats involved, and any
supporting tools or techniques.
2. Name concrete technologies: if the task mentions .xlsx files, one query should
be about spreadsheet/Excel manipulation. If it involves a specific framework,
name it.
3. Think about what SKILLS the agent needs, not the step-by-step procedure. Ask
'what would someone search for to find a how-to guide for this part of the task?'
4. Vary abstraction levels: include both specific tool queries (e.g. 'openpyxl
spreadsheet editing') and broader domain queries (e.g. 'data analysis with
Python').
5. Do NOT generate queries about generic file I/O, writing output files, or basic
Python operations; these are too broad to match any specific skill.
Respond with ONLY a JSON array of strings, no other text.
```

Figure 5: Multi-query prompt used by the retriever stage ($M > 1$).

Caching. Generated queries are cached to a JSON file keyed by task ID. This ensures identical queries are reused across experiment variants; for example, when comparing aggregation strategies, only the aggregation changes while the queries remain fixed.

Retry and Fallback. If multi-query JSON parsing fails, the system retries up to 2 times, increasing temperature by 0.3 per attempt. If all retries fail, it falls back to M independent single-query calls with incrementally increasing temperature (+0.3 per call) to encourage diversity.

A.3 Statistical Methodology

All confidence intervals are 95% bootstrap percentile intervals with 10,000 resamples over tasks (the primary unit of analysis), using a fixed random seed for reproducibility. For each task, the per-task Pass@k is computed from the 3 runs via the unbiased combinatorial estimator (Chen et al., 2021). We then bootstrap the mean across tasks to obtain CIs that reflect task-level variance, the dominant source of uncertainty with small benchmarks.

Pairwise significance is assessed via paired bootstrap tests: for each pair of conditions, per-task differences in Pass@k are resampled, yielding a two-sided p-value as the fraction of bootstrap samples whose mean difference crosses zero. To correct for multiple comparisons, we apply Holm-Bonferroni correction within each benchmark (3 comparisons on SkillsBench, 3 on Terminal-Bench), controlling the family-wise error rate. Effect sizes are reported as

Table 8: Example query generation outputs for the `court-form-filling` task. The task instruction is shown at the top, followed by the generated queries for $M = 1$ (single-query prompt) and $M = 5$ (multi-query prompt).

Task instruction: Fill the California Small Claims Court form at `/root/sc100-blank.pdf` based on the case description below, and save the filled one to `/root/sc100-filled.pdf`. Only fill in the necessary fields and leave the court-filled, optional fields or fields not mentioned in the case description below empty. Use this date format: xxxx-xx-xx. Case Description: I am Joyce He. It's my first time suing by small claims. I live in 655 S Fair Oaks Ave, Sunnyvale, CA 94086, my phone # is 4125886066, email: he1998@gmail.com. I want to sue Zhi Chen in 299 W Washington Ave, Sunnyvale, CA 94086. His phone #: 5125658878. He failed to return my security deposit of amount $1500 based on the signed roommate sublease contract after moving out. This situation happened from 2025-09-30 until 2026-01-19. I have asked him to return the money multiple times via text but he's not responding. The amount is listed on the signed roommate sublease contract. We both live in Sunnyvale, so I am filing where defendant lives. Please file it with date: January 19, 2026.	
$M = 1$	California Small Claims Court form filling automation using Python PDF libraries
$M = 5$	1. California Small Claims Court form filling 2. PDF form automation with Python 3. Filling out legal documents using PyPDF2 4. Using ReportLab for PDF generation in Python 5. Creating and editing PDF forms with pdfrw

Cohen's h for proportions, with standard thresholds ($|h| < 0.20$: negligible; < 0.50: small; < 0.80: medium).

For retrieval metrics (Experiment 3), CIs are computed analogously by bootstrapping per-task recall@k, precision@k, and MRR values across the 87 SkillsBench evaluation tasks.

A.4 Retriever Details

We compared several bi-encoder retriever models to determine which produces the best dense representations for skill retrieval. We also evaluated whether embedding only the skill description or the full skill content (description + SKILL.md body) leads to better retrieval. All models were evaluated on the SkillsBench oracle skill pairs with $k_1 = 1000$. Table 9 reports the results. The models evaluated are:

BAAI/bge-base-en-v1.5 (Xiao et al., 2023) is a general-purpose text embedding model from the BAAI General Embedding family with 110M parameters and a maximum sequence length of 512 tokens. It produces 768-dimensional embeddings and is trained on large-scale paired data with contrastive learning. If the input content exceeds 512 tokens, it is truncated to fit the context length.

BAAI/bge-m3 (Chen et al., 2024) is a multilingual, multi-functionality embedding model with 568M parameters and a maximum sequence length of 8192 tokens. It supports dense, sparse, and multi-vector retrieval within a single model, and is trained via self-knowledge distillation.

intfloat/e5-base-v2 (Wang et al., 2022) is a text embedding model with 110M parameters and a maximum sequence length of 512 tokens. It produces 768-dimensional embeddings and is trained with weakly-supervised contrastive pre-training on large-scale text pairs curated from the web. If the input content exceeds 512 tokens, it is truncated to fit the context length.

BM25 (Robertson & Zaragoza, 2009) is a sparse lexical retrieval method based on term frequency and inverse document frequency.

Table 9: Retriever model comparison on SkillsBench oracle skill retrieval. Models suffixed with "+ content" embed the full skill content instead of the description only. Bold indicates best per column. All metrics are averaged across tasks.

Retriever	Params	Metric	@1	@5	@10	@50	@100	@500	@1000	ms/query
bge-base	110M	R	**0.235**	**0.415**	**0.477**	**0.606**	**0.641**	**0.803**	**0.843**	22.8
		P	**0.471**	**0.207**	**0.121**	**0.032**	**0.017**	**0.004**	**0.002**	
		MRR				**0.553**				
bge-m3	568M	R	0.199	0.361	0.420	0.531	0.594	0.741	0.806	46.3
		P	0.391	0.166	0.105	0.027	0.015	0.004	0.002	
		MRR				0.479				
e5-base	110M	R	0.185	0.298	0.353	0.511	0.548	0.698	0.752	39.8
		P	0.379	0.145	0.086	0.027	0.015	0.004	0.002	
		MRR				0.449				
bge-m3 + content	568M	R	0.123	0.299	0.366	0.463	0.507	0.655	0.730	47.3
		P	0.241	0.133	0.089	0.023	0.012	0.003	0.002	
		MRR				0.358				
bge-base + content	110M	R	0.102	0.302	0.362	0.465	0.503	0.618	0.715	**22.1**
		P	0.172	0.133	0.085	0.024	0.013	0.003	0.002	
		MRR				0.304				
bm25	—	R	0.111	0.211	0.238	0.290	0.348	0.459	0.496	1592.1
		P	0.195	0.097	0.055	0.015	0.009	0.002	0.001	
		MRR				0.266				

BGE-base achieves the best performance across nearly all metrics while using fewer parameters (110M) and lower latency than larger alternatives. Embedding the description alone consistently outperforms embedding the full content for both BGE-base and BGE-M3, likely because skill descriptions are brief summaries optimized for semantic matching, whereas full content introduces noise from code examples and formatting. BM25 performs worst, suggesting that lexical matching is poorly suited to the semantic gap between task queries and skill descriptions. Based on these results, we select BGE-base with description-only embedding as the retriever for our pipeline.

Query Configuration. Using BGE-base, we further investigate how the number of generated queries (M) and the multi-query aggregation strategy affect retrieval performance. We evaluate two aggregation methods: **Reciprocal Rank Fusion (RRF)**, which merges per-query ranked lists using reciprocal rank scores, and **Union**, which takes the union of the top-k results from each query and re-ranks by the original cosine similarity. For Union aggregation, we also vary the per-query retrieval depth k (denoted tk). All experiments use the multi-query prompt (v2) and BGE-base with description-only embedding. Table 10 reports the results.

With RRF aggregation, a single query ($M = 1$) outperforms multi-query configurations at top-heavy metrics (R@1 = 0.209, MRR = 0.534), while multi-query variants ($M = 3, 5$) achieve higher recall at deeper cutoffs (R@1000 = 0.884 and 0.891, respectively). This suggests that RRF's rank-based fusion dilutes the signal of the best individual query at the top of the list, but broadens coverage at depth.

With Union aggregation, increasing M from 3 to 5 consistently improves top-heavy metrics (R@1 from 0.165 to 0.174, MRR from 0.464 to 0.487) while maintaining comparable deep recall. Varying the per-query depth tk has little effect on metrics up to R@100 but affects recall at deeper cutoffs. $tk = 200$ with $M = 5$ achieves the best R@1000 (0.905). Since the retriever is the first stage of the pipeline and its primary objective is to maximize recall, ensuring that relevant skills are not lost before the downstream reranking stages can refine the candidate set, we select Union aggregation with $M = 5$ and $tk = 200$, the configuration that achieves the highest R@1000 (0.905).

Modern retriever (recall ceiling). We compare the deployed `bge-base-en-v1.5` bi-encoder (110M) against `Octen-Embedding-8B` (8B), a top-ranked open-weight retriever on the RTEB

Table 10: Retriever query configuration comparison (BGE-base, description-only). M denotes the number of generated queries. tk denotes the per-query retrieval depth. Bold indicates best per column within each aggregation group. All metrics are averaged across tasks.

Agg.	M	tk	R@1	R@5	R@10	R@50	R@100	R@500	R@1000	MRR	ms/task
—	1	—	**0.209**	**0.417**	**0.491**	0.596	0.680	0.821	0.865	**0.534**	**21.5**
rrf	3	—	0.137	0.286	0.385	0.549	0.655	0.846	0.884	0.382	1597.5
rrf	5	—	0.146	0.292	0.407	0.586	0.664	0.837	0.891	0.384	2100.4
union	3	100	0.165	0.363	0.433	**0.606**	**0.687**	0.767	0.767	0.464	931.3
union	3	200	0.165	0.363	0.433	0.606	0.687	0.811	0.813	0.464	931.3
union	3	500	0.165	0.363	0.433	0.606	0.687	0.814	0.847	0.464	931.3
union	5	100	0.174	0.376	0.469	0.603	0.670	0.831	0.831	0.486	1945.1
union	5	200	0.174	0.376	0.469	0.603	0.670	**0.859**	**0.905**	0.487	1945.1
union	5	500	0.174	0.376	0.469	0.603	0.670	0.838	0.891	0.487	1945.1

Table 11: Modern retriever comparison on SkillsBench oracle skill retrieval, holding the generated query and pipeline fixed (87 tasks, 229 oracle skills).

Retriever (params)	R@5	R@10	R@100	R@1000	MRR
`BAAI/bge-base-en-v1.5` (110M, deployed)	0.376	0.469	0.670	0.905	0.487
`Octen/Octen-Embedding-8B` (8B)	**0.410**	**0.549**	**0.817**	**0.941**	**0.518**

and MTEB leaderboards, holding the generated query and the rest of the pipeline fixed (Table 11). The 8B model raises recall most at R@100 (0.670 → 0.817) and at the R@1000 ceiling the cascade inherits (0.905 → 0.941). Because embeddings are precomputed offline, the added cost is one-time reindexing and a larger index (4096-dim vs. 768-dim); we report the 8B as a recall ceiling and leave adopting a smaller Octen checkpoint as an efficiency-preserving deployment default to future work.

Structure-aware indexing. Table 12 reports the 3-way YAML/prose/code sectioned index (Section 4.5), encoded as three independent dense indices and late-fused at the score level. These numbers use a matched single-query setting to isolate sectioning against the desc-only bge-base baseline (R@1000 0.843), so they are not directly comparable to the deployed multi-query retriever (R@1000 0.905). Strategy A (3-way bge-base) improves recall at every cutoff, most at R@100, with the largest deep-recall gain from reciprocal-rank fusion (R@1000 0.843 → 0.925). The code-encoder controls (C1/C2, and Strategy B, which uses bge-code-v1 on the code section) underperform the general-purpose bge-base, because skill retrieval here is driven by natural-language descriptions rather than code.

In-/out-of-query oracle skills. To locate the source of the recall gap, we classify each oracle skill as *in-query* (its description shares a content word with the generated rerank query) or *out-of-query* (Table 13). The out-of-query share climbs roughly 10× from the top-5 bucket (4%) to the worst-ranked bucket (48%); within the worst-ranked bucket, the 23 oracle skills entirely missing from the top-1000 are 39% out-of-query. The oracle skills we miss are thus largely ones the task description never names (e.g. `hex-grid-spatial` for a Civilization 6 adjacency task), so the residual gap is partly query generation and motivates artifact-aware query generation (Section 5).

Stage-1 candidate count (k_1). We sweep the stage-1 cutoff $k_1 \in \{100, 250, 500, 1000, 2000, 4000\}$ (Table 14). Because the rerankers can only reorder and trim what stage 1 returns, k_1 sets the recall ceiling the rest of the cascade inherits. Recall rises steeply then saturates (0.670 at 100 to 0.892 at 1000, with only ∼4 more points by 2000 and flat thereafter), and a Kneedle elbow analysis places the knee at k_1=1000. Latency, not recall, is what makes 1000 the right operating point: of the ∼35s end-to-end (Table 5), only the dense retriever (3.3s) and shallow cross-encoder (1.6s) scale with k_1, while the deep reranker (25.6s) runs on a fixed top-100 and the selector on its top-10. Lowering k_1

Table 12: Structure-aware retrieval on SkillsBench (matched single-query setting). C1/C2 and Strategy B are code-encoder controls (bge-code-v1); Strategy A splits YAML/prose/code across three bge-base indices fused by RRF, max, mean, or sum-norm. Bold indicates best per column.

Variant	MRR	R@5	R@10	R@100	R@1000
desc-only (bge-base, baseline)	**0.553**	0.415	0.477	0.641	0.843
C2: desc + bge-code-v1	0.229	0.136	0.171	0.255	0.378
C1: content + bge-code-v1	0.097	0.085	0.092	0.126	0.226
A: 3-way (bge-base) · RRF	0.500	0.426	0.527	0.754	**0.925**
A: 3-way (bge-base) · max	0.535	0.467	**0.556**	0.740	0.914
A: 3-way (bge-base) · mean	0.522	0.443	0.511	0.691	0.922
A: 3-way (bge-base) · sum-norm	0.531	0.470	0.554	**0.773**	0.919
B: 3-way (mixed) · RRF	0.506	0.464	0.531	0.709	0.906
B: 3-way (mixed) · max	0.501	0.431	0.502	0.696	0.895
B: 3-way (mixed) · mean	0.172	0.069	0.162	0.714	0.909
B: 3-way (mixed) · sum-norm	0.528	**0.476**	0.534	0.704	0.896

Table 13: Oracle skills bucketed by rerank rank, with the fraction that are out-of-query (description shares no content word with the generated query). The three buckets partition all 229 oracle skills.

Rank bucket	# oracle skills	Out-of-query
Top-5 (rank 1–5)	97	4%
Top-100 (rank 6–100)	72	8%
Worst-ranked (rank $>$ 100 or missing)	60	48%

therefore trims at most the 1.6s shallow stage ($<$5% of the total) while giving up as much as 22 points of recall, so we size stage 1 for recall. (To reach 2000 and 4000 we reran retrieval with a deeper candidate pool, so recall@1000 reads 0.892 here versus 0.905 in the deployed configuration.)

A.5 Reranker Details

We evaluate cross-encoder reranker models and their configurations to select the best setup for the shallow and deep reranking stages. We first compare two cross-encoder models on the full 1000-candidate retriever output to determine model selection, then investigate the effect of input chunk size on the deep reranker operating on the 100-candidate shallow reranker output. The two cross-encoder models compared are:

cross-encoder/ms-marco-MiniLM-L-6-v2 (Reimers & Gurevych, 2019) is a lightweight cross-encoder with 6 Transformer layers and a maximum sequence length of 512 tokens, trained on the MS MARCO passage ranking dataset for fast inference.

BAAI/bge-reranker-v2-m3 (Chen et al., 2024) is a multilingual cross-encoder reranker with a maximum sequence length of 8192 tokens, trained with multi-stage knowledge distillation.

Model Comparison. We compare both models on the 1000-candidate retriever output with skill content truncated to 512 characters. We vary the number of queries ($M \in \{1, 3, 5\}$) and, for multi-query configurations, the score aggregation strategy: **mean** (average relevance score across queries), **max** (maximum score across queries), and **RRF** (reciprocal rank fusion). Single-query ($M = 1$) configurations require no aggregation. Table 15 reports the results.

BGE-reranker-v2-m3 with a single query ($M = 1$) achieves the best performance across nearly all metrics, with an MRR of 0.649, R@1 of 0.276, and R@50 of 0.720. However, BGE-

Table 14: Stage-1 candidate-count (k_1) sweep on SkillsBench retrieval (87 tasks, 229 oracle skills). Recall@k_1 is the cascade ceiling the downstream rerankers inherit.

Stage-1 cutoff k_1	**Recall@k_1 (cascade ceiling)**
100	0.670
250	0.793
500	0.838
1000	0.892
2000	0.932
4000	0.932

Table 15: Reranker model comparison on SkillsBench oracle skill retrieval. Both models are evaluated on the 1000-candidate retriever output with skill content truncated to 512 characters. M denotes the number of generated queries. Aggregation strategy applies only when $M > 1$. Bold indicates best per column. Underline indicates the selected configuration for the shallow reranker. All metrics are averaged across tasks.

Model	**Agg.**	M	**MRR**	**R@1**	**R@5**	**R@10**	**R@50**	**R@100**	**ms/task**
BGE-reranker-v2-m3	—	1	**0.649**	**0.276**	**0.465**	**0.557**	**0.720**	**0.786**	11999.3
ms-marco-MiniLM	—	1	0.587	0.259	0.460	0.520	0.687	0.776	**1135.2**
	max	3	0.438	0.148	0.333	0.438	0.634	0.687	3024.7
	max	5	0.433	0.129	0.354	0.436	0.651	0.759	5098.8
	mean	3	0.485	0.177	0.387	0.465	0.639	0.720	3024.7
	mean	5	0.518	0.191	0.400	0.471	0.659	0.737	5098.8
	rrf	3	0.449	0.169	0.324	0.416	0.643	0.754	3024.7
	rrf	5	0.429	0.162	0.352	0.414	0.664	0.762	5098.8

reranker-v2-m3 requires ∼12s per task to score 1000 candidates, approximately 10.6× the inference time of ms-marco-MiniLM (∼1.1s), making it impractical for the shallow reranking stage, which must score 1000 candidates in near real-time. The second-best configuration, ms-marco-MiniLM with $M = 1$ (underlined in Table 15), achieves competitive performance (MRR = 0.587, R@1 = 0.259) at a fraction of the cost. For both models, single-query scoring consistently outperforms multi-query configurations on top-heavy metrics, echoing the finding from the retriever stage (Section A.4). Among multi-query aggregation strategies, mean performs best, followed by max and RRF. We therefore select ms-marco-MiniLM-L-6-v2 with $M = 1$ for the shallow reranker (1000 → 100 candidates) and reserve BGE-reranker-v2-m3 for the deep reranker (100 → 10 candidates), where the smaller candidate set makes its higher computational cost acceptable.

Stronger and longer-context rerankers. On the same 1000-candidate retriever set (already at R@1000 0.905) and the same generated query, we evaluate stronger and longer-context cross-encoders (Table 16). R@100 is essentially flat across rerankers (0.776–0.786), so R@100 is retriever-bound and the reranker is not the recall bottleneck; a stronger cross-encoder mainly helps top-heavy ranking (bge-reranker-v2-m3 raises MRR from 0.587 to 0.649 and P@1 from 0.494 to 0.563 over ms-marco). Qwen3-Reranker-0.6B does not beat ms-marco at 512 chars and only matches bge's P@1 at 1024 chars, at higher latency and still without lifting R@100. The shallow cross-encoder is thus a latency/precision lever with little effect on recall, consistent with prior findings that a cross-encoder's recall contribution is bounded once the candidate set is fixed.

Deep Reranker Configuration. Using BGE-reranker-v2-m3 on the 100-candidate output of the shallow reranker, we investigate the effect of input chunk size, the character truncation length applied to skill content, on reranking quality. We evaluate chunk sizes of 1024, 2048, 4096, and 8192 characters, along with query count (M) and aggregation strategy. Table 17 reports results for all configurations.

Table 16: Stronger and longer-context rerankers on the same 1000-candidate retriever set and generated query. R@1000 (0.905) is fixed by the retriever candidate set. Bold indicates best per column.

Reranker (params)	Max Chars	R@5	R@10	R@100	R@1000	MRR	P@1
`cross-encoder/ms-marco-MiniLM-L-6-v2` (22.7M, deployed)	512	0.460	0.520	0.776	0.905	0.587	0.494
`BAAI/bge-reranker-v2-m3` (568M)	512	**0.465**	**0.557**	**0.786**	0.905	**0.649**	**0.563**
`Qwen/Qwen3-Reranker-0.6B` (596M)	512	0.420	0.552	0.777	0.905	0.541	0.414
`Qwen/Qwen3-Reranker-0.6B` (596M)	1024	0.451	0.530	0.766	0.905	0.629	**0.563**

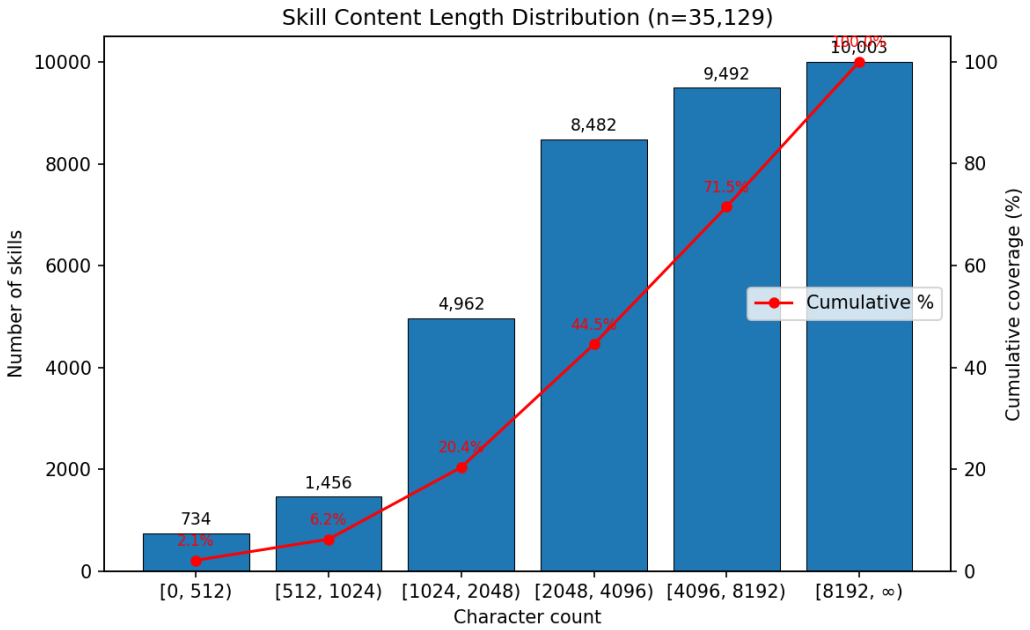

Figure 6: Distribution of skill content length (in characters) across the 35,129 indexed skills in the library. The red line shows cumulative coverage: a 4096-character window preserves 44.5% of skills in full.

Among single-query configurations, MRR decreases monotonically as chunk size increases (0.653 at 1024 to 0.620 at 8192), while R@5 and R@10 peak at 4096 characters (0.499 and 0.595, respectively). Skill content is long relative to these windows: as shown in Figure 6, a 4096-character truncation preserves only 44.5% of skills in full, and even 8192 characters reaches just 71.5%. Coverage alone therefore does not explain the optimum. Shorter truncations (1024 and 2048 characters, preserving 6.2% and 20.4% of skills) discard informative content and hurt R@5 and R@10, whereas extending to 8192 characters adds substantial coverage yet degrades every metric, indicating that the extra tail content dilutes the relevance signal more than it informs it. A 4096-character window balances the two: it captures the informative head of most skills without admitting the noisy remainder.

Multi-query configurations underperform single-query across all chunk sizes and both reranker models. Unlike the retrieval stage, where multiple queries broaden coverage over 35K skills, the reranking stage operates on a small, already-filtered candidate set where a single focused query is sufficient to discriminate among candidates and additional queries likely introduce more noise than signal.

Based on these results, we select BGE-reranker-v2-m3 with 4096-character truncation and a single query ($M = 1$) as the deep reranker configuration, as it achieves the best R@5 (0.499) and R@10 (0.595), the metrics most relevant for the deep reranker's output size of $k_3 = 10$, while maintaining strong MRR (0.634).

Table 17: Deep reranker configuration comparison (BGE-reranker-v2-m3) on the 100-candidate shallow reranker output. Chunk size denotes the character truncation length applied to skill content. Bold indicates best per column. All metrics are averaged across tasks.

Chunk Size	Agg.	M	MRR	R@1	R@5	R@10	ms/task
1024	—	1	**0.653**	0.270	0.476	0.568	**2988.7**
1024	max	3	0.517	0.166	0.420	0.537	7894.1
1024	max	5	0.552	0.197	0.439	0.536	27875.1
1024	mean	3	0.558	0.198	0.429	0.545	7894.1
1024	mean	5	0.573	0.208	0.472	0.539	27875.1
1024	rrf	3	0.505	0.194	0.413	0.519	7894.1
1024	rrf	5	0.510	0.185	0.418	0.512	27875.1
2048	—	1	0.637	**0.272**	0.490	0.585	5937.5
2048	max	3	0.521	0.175	0.418	0.525	18379.1
2048	max	5	0.535	0.183	0.433	0.518	29338.4
2048	mean	3	0.557	0.219	0.460	0.556	18379.1
2048	mean	5	0.580	0.232	0.447	0.545	29338.4
2048	rrf	3	0.478	0.164	0.389	0.530	18379.1
2048	rrf	5	0.466	0.147	0.408	0.505	29338.4
4096	—	1	0.634	0.271	**0.499**	**0.595**	12773.9
4096	max	3	0.522	0.172	0.443	0.544	37757.2
4096	max	5	0.529	0.183	0.412	0.541	129426.4
4096	mean	3	0.532	0.205	0.463	0.537	37757.2
4096	mean	5	0.587	0.246	0.431	0.566	129426.4
4096	rrf	3	0.458	0.159	0.380	0.526	37757.2
4096	rrf	5	0.489	0.197	0.400	0.532	129426.4
8192	—	1	0.620	0.251	0.467	0.553	30462.3
8192	max	3	0.509	0.169	0.402	0.552	95319.6
8192	max	5	0.506	0.178	0.401	0.523	154890.4
8192	mean	3	0.534	0.220	0.421	0.530	95319.6
8192	mean	5	0.547	0.227	0.425	0.530	154890.4
8192	rrf	3	0.468	0.179	0.392	0.492	95319.6
8192	rrf	5	0.467	0.181	0.400	0.475	154890.4

A.6 Selector Details

The selector is a two-step LLM-based filter that operates on the top-k candidates (default k=10) from the deep reranker. It determines which retrieved skills, if any, should be injected into the agent's workspace.

Step 1: Relevancy Filter. An LLM (gpt-4o-mini) receives the task instruction and each candidate skill's full `SKILL.md` content. It makes a binary judgment: is this skill topically relevant to the task? The LLM outputs a JSON list of selected candidate indices. Skills that are off-topic or unrelated are removed. The system instruction (Figure 7) and user instruction (Figure 8) structure this as a classification task.

Step 2: Specificity Filter. Skills that pass relevancy are evaluated for actionable specificity. A second LLM call (same model) assesses whether the skill provides domain-specific knowledge that would save the agent meaningful exploration time, as opposed to generic methodology, checklists, or information the agent can derive from its training data or the task environment. The specificity instruction (Figure 9) defines the INJECT/REJECT criteria.

Table 18: No-selector ablation on SkillsBench (Codex, GPT-5-mini). Deep-reranker-only injects the deep reranker's top-10 skills directly; full SkillFlow adds the LLM selector. Paired bootstrap p-values vs. full SkillFlow are reported in the text.

Pipeline	Pass@1	Cost/Task
No Skills	9.2 [4.1, 14.9]	\$0.026
SkillFlow (with selector)	16.4 [9.7, 23.6]	\$0.025
Deep-reranker top-10 (no selector)	15.9 [9.7, 23.1]	\$0.026

No-selector ablation. To test whether the selector is necessary, we compare full SkillFlow against injecting the deep reranker's top-10 skills directly, with no selector (Table 18). Downstream Pass@1 is statistically indistinguishable (16.4% vs. 15.9%, paired bootstrap $p = 0.91$) and cost is unchanged (\$0.025 vs. \$0.026, $p = 0.50$), while both remain well above the no-skill baseline (9.2%). The selector's role is therefore context compaction, shrinking the injected set from 10 to a mean of 2.8 skills (Table 2), rather than accuracy, so we treat it as an optional gate (Section 3, Section 4.3).

A.7 Vercel Baseline Details

We compare against skills.sh, Vercel's public skill search API, as a commercial baseline. To ensure a fair comparison, we reuse the same LLM-generated queries from SkillFlow's cache. For each task, the cached query is sent to the skills.sh REST API (`GET https://skills.sh/api/search?q=<query>`), and the top-1 non-duplicate result is selected. Each matched skill is then downloaded via Vercel's official CLI (`npx skills add <source> -s <skillId> -y --copy -a claude-code`), which fetches the full skill folder from the source GitHub repository. Downloaded skills are evaluated using the same Harbor-based harness, agent model, and concurrency settings as all other conditions, with 1:1 task-to-skill matching by task name.

A.8 Task Exclusion Criteria

We exclude tasks from both benchmarks when they cannot be evaluated reliably in our sandboxed Docker environment. Exclusion decisions were made during pilot runs based on three criteria: (1) **infrastructure** errors, where the Daytona cloud sandbox failed to provision the task environment or the Docker build failed (e.g., OOM during model loading, intermittent downloads, hardcoded host paths), with Docker build failures documented on the SkillsBench website; (2) **reproducibility** issues, where the task's oracle verifier produces intermittent failures due to environment-sensitive tests, flaky builds, or external API dependencies, also documented on the SkillsBench website; and (3) **resource constraints**, where the task requires large corpus downloads or model training exceeding the sandbox budget. Table 19 lists all excluded tasks.

A.9 Qualitative Case Studies

Case Study 1: Invoice Fraud Detection.

Task. Injected skills: `fuzzy-match`, `pdf`, `xlsx`. Outcome: 2/2 tests passed.

Agent behavior. The dense retriever returns generic PDF extraction skills, none directly relevant to fraud detection. The cross-encoder reranker surfaces `fuzzy-matching` (a community skill) to rank 1 with a score of 3.74, pushing it ahead of 689 candidates. The deep reranker confirms this ranking (0.911) and the LLM selector marks it as both relevant and specific. The agent used all three injected skills (`fuzzy-match`, `pdf`, `xlsx`) and passed all tests. This case illustrates how reranking transforms an uninformative Stage 1 ranking into actionable skill selection.

Selector System Prompt (Relevancy)

```
You are a skill utility judge for an AI coding agent. The agent will receive at most ONE candidate "skill"
(an instruction document) to help it solve a task. Your job is to decide which single skill, if any, is
worth injecting.

IMPORTANT: Injecting skills has a COST: approach bias and distraction. The agent solves most tasks BETTER
without skills. Only inject a skill if you are confident it will help more than it hurts.

## Two-phase evaluation

### Phase 1: Does the agent need external help?

Read the task description and ask:

"Does this task require specific procedural knowledge that can only come from hands-on experience with
the exact tools/environment, or can the agent solve it from first principles?"

The agent is an expert-level AI with deep knowledge of programming, algorithms, frameworks, and
well-documented tools.

The agent does NOT need help when:
- The task has a clear logical or algorithmic solution, even if the domain sounds specialized.
- The task involves well-documented tools or formats.
- The task description specifies what to do clearly enough that reasoning is sufficient.

The agent DOES need help when:
- The task requires environment-specific procedural knowledge: exact command sequences, build system
workarounds, boot sequences, or binary format internals.
- The task involves tool interactions with non-obvious failure modes: undocumented behavior,
version-specific bugs, or subtle compatibility issues.

If the agent likely does NOT need help → select nothing.

### Phase 2: Pick the single best skill (only if the agent needs help)

From the candidates, select at most one: the single skill that most precisely matches the task's core
difficulty.

KEEP a skill only if:
1. It addresses the exact difficulty identified in Phase 1.
2. It provides concrete, non-obvious procedural details (exact commands, specific parameter values,
workarounds).
3. It is precisely aligned with the task, not a generic guide for a related topic.

REJECT a skill if:
1. It restates knowledge the agent already has.
2. It provides general methodology rather than specific procedures.
3. It could mislead the agent if task details differ from the skill's assumptions.

## Output

Return a JSON object: {"selected": [N]} where N is the single best candidate number, or {"selected": []}
if no skill is worth injecting.
```

Figure 7: System prompt for the selector relevancy filter (Step 1). The LLM classifies each candidate skill as relevant or irrelevant to the task.

Selector User Prompt (Shared)

```
## Task
{{ query }}

## Candidate Skills
{% for candidate in candidates %}
### Candidate {{ loop.index }}
{{ candidate.content or candidate.description }}
{% endfor %}
```

Figure 8: User prompt template shared by both selector steps. The task instruction and candidate skill contents are populated via Jinja2 templating.

Selector Specificity Prompt (Step 2)

```
You are a second-pass filter for AI agent skill injection. The candidate skill below has already been
judged as topically relevant. Your job is to determine whether injecting it would save the agent
meaningful time.

## Context

The agent is an expert-level AI coding assistant. It has broad knowledge of programming languages,
frameworks, and common tools. It can also explore the task environment: reading files, running commands,
and checking documentation.

However, the agent works under a time budget. Skills are valuable when they reduce the exploration and
trial-and-error the agent would otherwise need.

## Decision criteria

### INJECT: the skill provides domain-specific knowledge that would take the agent significant effort to
assemble on its own:

- Specific API usage patterns, method signatures, or library workflows for specialized tools
- Domain-specific configuration, parameters, or data formats
- Multi-step procedures that require combining knowledge from multiple sources
- Tool-specific idioms or integration patterns not obvious from --help alone
- Concrete examples, code snippets, or templates for niche tasks

### REJECT: the skill adds no time savings:

- Pure methodology or approach guidance without concrete implementation details
- Checklists or best-practice lists without actionable specifics
- Content that merely restates or reorganizes the task description
- Vague overviews that lack concrete code, commands, or configurations

When in doubt, INJECT. A marginally useful skill is better than a missed one.

## Output

Return {"selected": [N]} if the skill passes, or {"selected": []} if it does not.
```

Figure 9: Specificity filter prompt (Step 2). Evaluates whether a relevant skill provides actionable, domain-specific knowledge (INJECT) or only generic guidance (REJECT).

Table 19: Excluded benchmark tasks. Of 87 SkillsBench tasks, 22 are excluded (65 retained). Of 89 Terminal-Bench tasks, 1 is excluded (88 retained).

Benchmark	Task	Category	Reason
SkillsBench	earthquake-phase-association	Infrastructure	Daytona sandbox error
SkillsBench	fix-druid-loophole-cve	Infrastructure	Daytona sandbox error
SkillsBench	fix-erlang-ssh-cve	Infrastructure	Daytona sandbox error
SkillsBench	latex-formula-extraction	Infrastructure	Daytona sandbox error
SkillsBench	organize-messy-files	Infrastructure	Daytona sandbox error
SkillsBench	parallel-tfidf-search	Infrastructure	Daytona sandbox error
SkillsBench	quantum-numerical-simulation	Infrastructure	Daytona sandbox error
SkillsBench	setup-fuzzing-py	Infrastructure	Daytona sandbox error
SkillsBench	shock-analysis-demand	Infrastructure	Daytona sandbox error
SkillsBench	syzkaller-ppdev-syzlang	Infrastructure	Daytona sandbox error
SkillsBench	taxonomy-tree-merge	Infrastructure	Daytona sandbox error
SkillsBench	video-filler-word-remover	Infrastructure	Daytona sandbox error
SkillsBench	video-tutorial-indexer	Infrastructure	Daytona sandbox error
SkillsBench	multilingual-video-dubbing	Infrastructure	TTS model download fails intermittently during Docker build
SkillsBench	scheduling-email-assistant	Infrastructure	Docker compose mounts hardcoded host path
SkillsBench	speaker-diarization-subtitles	Infrastructure	Whisper model loading triggers OOM during Docker build
SkillsBench	dynamic-object-aware-egomotion	Reproducibility	Oracle outputs non-serializable numpy types
SkillsBench	fix-build-google-auto	Reproducibility	Maven build flaky under Docker networking
SkillsBench	pedestrian-traffic-counting	Reproducibility	Oracle depends on external vision API keys
SkillsBench	r2r-mpc-control	Reproducibility	MPC settling time sensitive to Docker CPU scheduling
SkillsBench	reserves-at-risk-calc	Reproducibility	Numerical precision varies across platforms
SkillsBench	simpo-code-reproduction	Reproducibility	Build timeout; passes with extended timeout
Terminal-Bench	train-fasttext	Resource	Large corpus download and model training

Table 20: Pipeline progression for `invoice-fraud-detection`.

Stage	Skill	Score	GT?
Stage 1: Dense Retriever	`pdf-extractor-1`	0.811	
	`pdfco`	0.766	
	`llmwhisperer`	0.748	
	`pandas-best-practices`	0.746	
	`processing-data`	0.745	
Stage 2: Cross-Encoder	`fuzzy-matching`	3.743	
	`transaction-classification-debugger`	1.156	
	`fuzzy-match`	0.025	✓
	`fuzzy-match`	0.025	
	`nucleo-matcher`	-1.767	
Stage 3: Deep Reranker	`fuzzy-matching`	0.911	
	`fuzzy-match`	0.829	✓
	`fuzzy-match`	0.829	
	`transaction-classification-debugger`	0.600	
	`nucleo-matcher`	0.285	
Stage 4: LLM Selector	`fuzzy-matching`	relevant	
	`fuzzy-match`	filtered	✓
	`fuzzy-match`	filtered	
	`transaction-classification-debugger`	filtered	
	`nucleo-matcher`	filtered	

Case Study 2: Court Form Filling.

Task. Injected skills: `pdf`. Outcome: 0/5 tests passed.

Agent behavior. All pipeline stages surface PDF-related skills: `python-pdf` is the top candidate by Stage 3 (score 0.706) and the only skill the selector marks as relevant. The agent used the injected `pdf` skill, which provides general PDF manipulation guidance (e.g., PyPDF2 API usage). However, the task requires filling specific fields in a California Small Claims Court form (SC-100), a domain-specific mapping that no generic PDF skill can provide. All 5 tests failed. This case illustrates the "skill quality is the bottleneck" finding:

Table 21: Pipeline progression for court-form-filling.

Stage	Skill	Score	GT?
Stage 1: Dense Retriever	pdf-4	0.761	
	python-pdf	0.754	
	writing-tests-4	0.749	
	fixture-table	0.745	
	pdf-19	0.740	
Stage 2: Cross-Encoder	pdf	-0.736	
	pdf-33	-0.736	
	pdf-30	-0.736	
	pdf-28	-0.736	
	pdf-26	-0.987	
Stage 3: Deep Reranker	python-pdf	0.706	
	pdf-16	0.698	
	pdf-20	0.687	
	pdf-7	0.683	
	pdf-21	0.683	
Stage 4: LLM Selector	python-pdf	relevant	
	pdf-16	filtered	
	pdf-20	filtered	
	pdf-7	filtered	
	pdf-21	filtered	

the pipeline correctly identifies the technology domain (PDF manipulation) but the library lacks a sufficiently specific skill for legal form filling.

Case Study 3: Gravitational Wave Detection.

Task. Injected skills: conditioning, matched-filtering. Outcome: 6/9 tests passed.

Agent behavior. Retrieval is near-perfect: the oracle skill matched-filtering ranks first at every stage (scores: $0.844 \rightarrow 7.51 \rightarrow 0.999$), and conditioning ranks second throughout. However, the LLM selector marks *all* 10 candidates as not relevant (relevancy=0), including both oracle skills. Despite this, the agent used both injected skills and passed 6 of 9 tests, failing on the most complex signal processing subtasks. This case highlights two issues: (1) the LLM selector can be overly conservative with domain-specific terminology, and (2) even with perfect retrieval and skill adoption, execution complexity limits performance.

A.10 Retrieved-Skill Usage

We analyze how agents use retrieved skills across the SkillsBench runs, separating oracle from non-oracle retrievals (Table 23). Non-oracle skills make up 43% of the injected set but are opened only 17% of the time, versus 42% for oracle skills, about half as often. Relative to the no-skill baseline, retrieved skills improve 16 of 64 tasks and regress 4 (per-task mean Pass over three runs, paired bootstrap $p = 0.02$), and the improvement concentrates

Table 22: Pipeline progression for `gravitational-wave-detection`.

Stage	Skill	Score	GT?
Stage 1: Dense Retriever	`matched-filtering`	0.844	✓
	`conditioning`	0.757	✓
	`gwosc`	0.694	
	`scipy-best-practices`	0.656	
	`tuning-hyperparameters`	0.650	
Stage 2: Cross-Encoder	`matched-filtering`	7.510	✓
	`conditioning`	4.322	✓
	`gwosc`	-4.145	
	`large-cell-ratio-matching`	-5.281	
	`detect-python-command`	-6.234	
Stage 3: Deep Reranker	`matched-filtering`	0.999	✓
	`conditioning`	0.989	✓
	`gwosc`	0.270	
	`bio-filter-sequences`	0.156	
	`oe-keyword-audit`	0.094	
Stage 4: LLM Selector	`matched-filtering`	filtered	✓
	`conditioning`	filtered	✓
	`gwosc`	filtered	
	`bio-filter-sequences`	filtered	
	`oe-keyword-audit`	filtered	

Table 23: Retrieved-skill usage on SkillsBench (per-task mean Pass over three runs). Rows break tasks down by what SkillFlow injected; Helped/Hurt are per-task counts relative to the no-skill baseline. Oracle vs. non-oracle load rates are reported in the text.

Injected	Tasks	SkillFlow	Baseline	Helped	Hurt
All tasks	64	0.167	0.094	16	4
Oracle present	55	0.194	0.103	16	3
Non-oracle only	4	0.000	0.000	0	0
No skills	5	0.000	0.067	0	1

in tasks where retrieval recovered at least one oracle skill. In the four tasks where only non-oracle skills were retrieved, the agent loads them but solves none, and the clearest regression (`jax-computing`) is one where the agent fixates on a non-oracle skill and never opens the oracle skill. Non-oracle skills are thus mostly not useful as-is but rarely degrade performance, which motivates skill adaptation as future work.

A.11 Oracle Ceiling Analysis

To understand why even perfect retrieval yields only 19.5% Pass@1 on SkillsBench, we examine three converging factors evident in our results. First, *agent non-adoption*: despite oracle skills being available for every task, agents use at least one skill in only 69.2% of tasks, meaning 30.8% of tasks receive no skill benefit at all because the agent judges the provided skills as unnecessary. This likely reflects a mismatch between the skills' instructional style (general-purpose reference documents) and what the agent recognizes as actionable for a

specific task instance. Second, *task inherent difficulty*: the no-skill baseline itself stands at only 9.2% Pass@1, confirming that SkillsBench tasks are challenging for the agent model regardless of skill availability. Many tasks involve multi-step workflows (e.g., filling legal PDF forms, configuring complex build systems) where a single misstep in tool use or reasoning cascades into failure, and a reference document alone cannot compensate for these execution errors. Third, *low consistency across runs*: the oracle condition achieves Pass@3 of 35.4% but Pass^3 of only 6.2%, indicating that most oracle-aided successes are non-deterministic: the agent solves the task in some runs but not others. This high variance suggests that even when skills do help, the benefit is fragile and depends on the specific reasoning trace the agent follows.

Together, these factors delineate a ceiling imposed not by retrieval quality but by the interaction between skill format, agent reasoning, and task complexity. Closing this gap will require advances on multiple fronts: skills that provide more directly executable artifacts rather than reference prose, agents that can more reliably incorporate instructional content into their planning, and evaluation protocols that account for the stochastic nature of agent execution.