SKILLDIVER: CONSTRAINT-GUIDED GENERATION AND ADAPTIVE DIFFERENTIAL TESTING TO EVALUATE SKILL SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly use reusable skills, but selecting relevant skills becomes less reliable as skill libraries grow. Evaluating skill selection at scale remains challenging for two reasons: extending labeled benchmarks to new skills and usage scenarios requires costly task construction and label validation, while a task may admit multiple valid skill selections, making per-task correctness labels difficult to define. We develop SKILLDIVER, a framework that combines constraint-guided task generation with adaptive differential testing to evaluate skill selection without requiring per-task correctness labels. Constraint-guided generation extracts and samples fine-grained constraints from skill documents to generate tasks. Adaptive differential testing treats the selection from the full skill library as the selection under test and assesses it by comparing with selections from smaller reference pools. We use two types of reference pools: the target pool, which contains the skills used to generate the task, and the adaptive pool, which additionally includes skills in the selection under test. When the selection under test is not reproduced in either reference execution, SKILLDIVER reports a selection deviation as a potential selection error. Across two agents and four settings, SKILLDIVER exposes more selection deviations than GoS-Random-Direct, increasing the selection deviation rate by 9.8% to 90.4%. For validation on four labeled benchmarks, the differential module achieves an average F1 of 84.58% in identifying selections that disagree with benchmark labels, outperforming an LLM judge by 28.46 percentage points. With skill chains held fixed, constraint-guided generation increases within-chain semantic diversity by 76.6% on SkillsBench and 61.6% on R3-Skill. Overall, SKILLDIVER explores how task generation and differential testing can support automated skill selection testing without requiring per-task correctness labels.

1 INTRODUCTION

Reusable skills give LLM agents specialized instructions, scripts, and resources that can be loaded on demand (Anthropic, 2025; Zhou et al., 2026; Xu & Yan, 2026). For each task, the agent selects relevant skills from a skill library, but this selection becomes less reliable as the library grows (Li, 2026; Zheng et al., 2026b; Kang et al., 2026). Existing benchmarks assess skill selection against predefined labels (Kang et al., 2026; Su et al., 2026; Wang et al., 2026), which are costly to extend to new skills and scenarios and may overlook cases where multiple selections are valid. Automated agent testing offers a complementary evaluation paradigm that generates inputs beyond fixed test cases and evaluates agent behavior through execution (Guo et al., 2026; Ruan et al., 2024).

Evaluating skill selection in this way requires both diverse tasks and a reliable criterion for judging skill selection. Skill documents often describe multiple supported behaviors and the conditions under which they apply (Tan et al., 2026). However, prompting an LLM with an entire skill document may produce tasks concentrated on a few documented uses. As a result, a large number of generated tasks may still cover only a small subset of a skill's documented capabilities. For example, in our task diversity experiment, all five tasks generated directly from `sympy` involve equation solving, calculus, or algebraic manipulation, although the skill also supports physics, geometry, and code generation (Table 14). A reliable criterion for judging skill selection is also difficult to define. A

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

generated task may be supported by multiple skills, so the skill used to generate it is not necessarily its unique correct label. Other skills or the agent's existing capabilities may support the same request (Su et al., 2026). For example, a task generated from `FFmpeg Audio Processing` asks for audio-format conversion. OpenHands selects `FFmpeg Format Conversion` from the available library. Both skills support audio-format conversion, so treating the source skill as the only correct selection would incorrectly mark a valid alternative as incorrect (Appendix E.3).

We introduce SKILLDIVER, which combines constraint-guided task generation with adaptive differential testing to evaluate skill selection. For each generated task, the selection made from the full pool is treated as the selection under test, with the full pool representing the skill library available to the agent. The generator decomposes skill documents into fine-grained constraints and samples constraints to generate diverse task scenarios. The observed degradation of skill selection accuracy with increasing skill pool size motivates us to exploit this size sensitivity as a differential signal for identifying potentially inappropriate selections. The differential module repeatedly runs the same task in smaller reference pools to determine whether the selection under test can be reproduced, without requiring a correctness label for each generated task. We use two types of reference pools. The target pool contains only the skills used to generate the task. If the selection under test contains skills outside the target pool, the adaptive pool augments the target pool with the selection under test, allowing us to check whether the selection under test may be a valid alternative. We define a selection deviation only when the selection under test is not observed in either the target or adaptive reference executions. After each testing round, SKILLDIVER updates sampling weights to favor skills with fewer prior tests or higher observed deviation rates.

We evaluate SKILLDIVER on deviation discovery, differential judgment reliability, and task diversity. Across Codex (OpenAI, 2025) and OpenHands (Wang et al., 2025) in four settings, SKILLDIVER exposes more selection deviations than GoS-Random-Direct under the same test-case budget. The relative increase in selection deviation rate ranges from 9.8% to 90.4%, while SKILLDIVER also covers more skills and identifies more skills with observed selection deviations. On four labeled benchmarks, the differential module achieves an average F1 score of 84.58% in identifying selections that disagree with benchmark labels, outperforming an LLM judge baseline by 28.46 percentage points. Human validation further supports the practical relevance of the detected deviations: the two annotators judged 77.0% and 79.5% of the reported selections incorrect, respectively. Moreover, when generating tasks from the same skills, constraint-guided generation improves within-chain semantic diversity by 76.6% on SkillsBench and 61.6% on R3-Skill.

We summarize our contributions as follows: (1) We develop constraint-guided task generation that extracts and samples fine-grained constraints from skill documents to generate diverse tasks. (2) We propose adaptive differential testing that assesses skill selection by comparing agent executions across skill pools, without requiring per-task correctness labels for generated tasks. (3) We implement these designs in SKILLDIVER and use feedback from differential testing to prioritize skill chains with fewer prior tests or higher observed deviation rates for subsequent task generation. (4) We evaluate SKILLDIVER across two agents and four test settings, with benchmark and human validation of differential judgments.

2 TESTING SKILL SELECTION

This section defines skill selection from execution traces and introduces differential testing for assessing it. Figure 1 provides a running example of these definitions and the testing procedure.

Problem Formulation. We aim to uncover potential selection deviations when the agent selects skills from a realistic skill library to handle user tasks. Let $\mathcal{S}$ be the skill library and A the agent under test. For a task x and a skill pool $P \subseteq \mathcal{S}$, let $\mathrm{Sel}(A, x, P)$ denote the set of skills selected in the execution trace of A. The selection under test is $Y_{\text{full}} = \mathrm{Sel}(A, x, P_{\text{full}})$, where the full pool P_{full} represents the realistic skill library including all skills available to the agent. If the execution selects no skill, then $\mathrm{Sel}(A, x, P) = \varnothing$. We use differential testing to compare the selection under test with reference selections obtained by rerunning the same task in smaller reference pools.

Challenges in testing skill selection. Skill documents may describe multiple task scenarios and the conditions under which they apply. The document shown on the left of Figure 1 describes two such scenarios. One checks that an NPZ file contains an entry for every sampled frame. The other

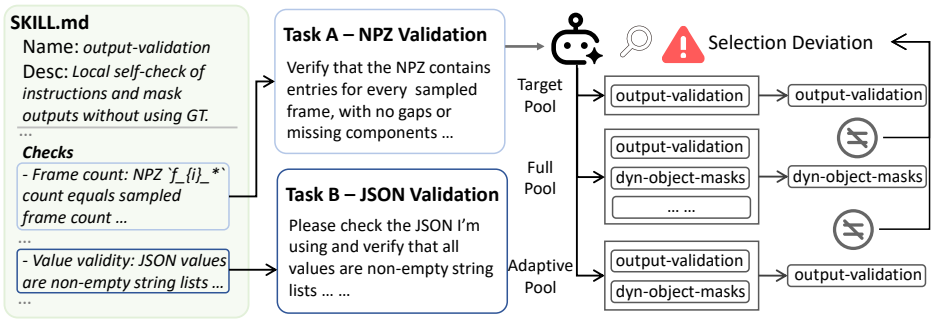

Figure 1: An `output-validation` skill describes two scenarios: NPZ frame coverage and JSON value checks (left). The pool comparison for Task A is shown on the right. The target-pool executions select `output-validation`, while the full-pool execution selects `dyn-object-masks`. The adaptive pool additionally includes `dyn-object-masks`, but the adaptive-pool executions still select only `output-validation`, so SKILLDIVER reports a selection deviation.

checks that JSON values are non-empty string lists. However, prompting the entire document may repeatedly generate the NPZ check while leaving the JSON check untested. Additionally, the skills used to construct a task do not necessarily determine a unique valid selection. Another skill or the agent's existing capabilities may support the same task. Thus, other selections may also be valid.

Test design. These challenges motivate two parts of our test design. The first addresses incomplete scenario coverage. Constraint-guided generation extracts fine-grained uses and conditions from a skill document and samples them to generate tasks for different documented scenarios. In Figure 1, the two checks described in `output-validation` are instantiated as Task A for NPZ frame coverage and Task B for JSON value validation. The second addresses the lack of a unique expected selection. To assess the selection under test without per-task correctness labels, we employ differential testing. Differential testing is often used to identify potential errors by comparing the outputs of systems on the same input (Etemadi et al., 2025; McKeeman, 1998). Here, we adapt this idea by holding both the task and agent fixed, while varying the available skill pool and comparing the resulting skill selections. Specifically, to assess the selection under test (e.g., `dyn-object-masks` in Figure 1), adaptive differential testing uses two types of reference pools. The target pool P_{tar} contains the skills used to generate the task (e.g., `output-validation` in Figure 1), while the adaptive pool P_{adp} augments the target pool with `dyn-object-masks`. The reference selections from both pools contain only `output-validation`. Since neither reference pool reproduces the selection under test, SKILLDIVER reports a selection deviation.

3 THE SKILLDIVER FRAMEWORK

Figure 2 shows the overall workflow of SKILLDIVER. Before testing, Constraint Extraction converts each skill document in the skill library into constraint groups. In each round, Constraint-guided Task Generation selects a skill chain, samples one constraint group per skill, and combines the samples into a task. Adaptive Differential Testing runs the task in the target and full pools, followed by the adaptive pool when the full-pool execution selects additional skills. Feedback-guided Scheduling uses the comparison results to update the weights for subsequent chain selection. We repeat these steps until the test-case budget is exhausted.

3.1 CONSTRAINT EXTRACTION

A skill document may support multiple distinct uses. Directly using the entire document to generate can lead to repetitive task generation, as the model may repeatedly generate tasks centered around the same or similar use. Therefore, the generator should distinguish among the documented uses before constructing a task. We represent each use together with the conditions under which it applies and sample these representations during generation.

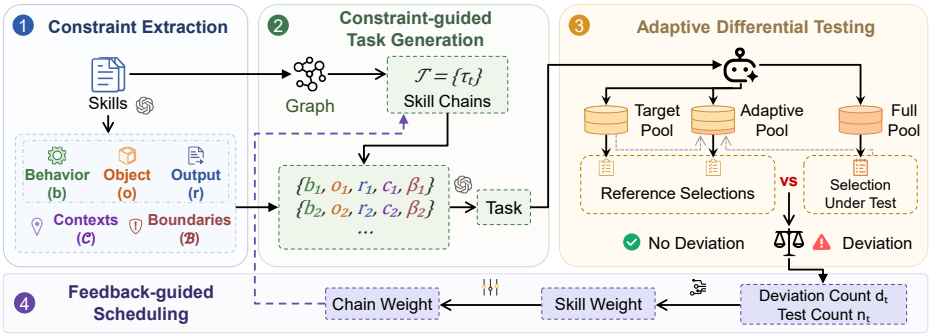

Figure 2: Overview of SKILLDIVER. It extracts constraints from skill documents, generates tasks from skill chains, and compares the selection under test with reference selections across skill pools. The comparison guides subsequent skill sampling.

A constraint group is defined as $g = (b, o, r, \mathcal{C}, \mathcal{B})$. The behavior b, object o, and output r define a core intent: the requested action, its target, and the expected result, respectively. The context candidates $\mathcal{C}$ describe concrete situations in which that intent applies, while the boundary candidates $\mathcal{B}$ describe situations that should be excluded from that intent. For each skill $s \in \mathcal{S}$, the resulting constraint groups are denoted by $\mathcal{G}(s) = \{g_{s,1}, \ldots, g_{s,m_s}\}$. Each group associates an intent with its own conditions, preventing the sampler from mixing conditions across unrelated intents. To construct constraint groups, an LLM first extracts core intents from the skill document. We extract the behavior, object, and output together to reduce the risk of combining fields from different intents. For each extracted intent, a second LLM call identifies its context and boundary candidates while keeping the intent fixed. The extraction prompt uses the skill description to define its selection scope and the document body to clarify the supported intents. Appendix F.3 shows the recorded groups, and Appendix F.2 gives the extraction prompts and response schemas.

3.2 CONSTRAINT-GUIDED TASK GENERATION

Task generation uses the extracted constraint groups. For each task, SKILLDIVER selects a candidate skill chain, samples one group per skill, and combines the sampled constraints into a task.

Candidate skill chains. A task may span several steps or operations, requiring multiple skills to work together (Wang et al., 2026; Gao, 2026). However, arbitrarily combining skills may produce requests in which their uses are not meaningfully related. SKILLDIVER uses the dependency and workflow relations in Graph of Skills (GoS) (Liu et al., 2026) to construct a skill relation graph and extract candidate chains. We represent each candidate chain as $\tau = \{s_1, \ldots, s_k\} \subseteq \mathcal{S}$. The resulting set $\mathcal{T}$ contains the candidate chains considered for task generation. In each round, the scheduler selects one chain from $\mathcal{T}$ according to the weights in Section 3.4.

Constraint sampling. Given a selected skill chain τ, SKILLDIVER samples one constraint group $g_s = (b_s, o_s, r_s, \mathcal{C}_s, \mathcal{B}_s)$ from $\mathcal{G}(s)$ for each skill $s \in \tau$. It retains the complete core intent (b_s, o_s, r_s) and samples one context $c_s \in \mathcal{C}_s$ and one boundary $\beta_s \in \mathcal{B}_s$, when available. The generation input for the chain is defined as $\Gamma(\tau) = \{(b_s, o_s, r_s, c_s, \beta_s) \mid s \in \tau\}$. Sampling different constraint groups across rounds varies the intents and conditions used to generate tasks for the same skill.

Task generation. Given $\Gamma(\tau)$, an LLM generates a user request (i.e., task) x based on it. The generated task expresses the sampled operations and expected results within their corresponding contexts while excluding the situations specified by the sampled boundaries. It may include additional details consistent with these constraints, but must not introduce unrelated requirements. Appendix F.2 gives the generation prompt and runtime inputs.

3.3 ADAPTIVE DIFFERENTIAL TESTING

Evaluating skill selection for generated tasks requires assessing the selection under test without per-task correctness labels. The source skills alone cannot directly serve as correctness labels because other skills or the agent's own capabilities may also support the task. Prior studies show that skill

selection becomes less reliable as skill libraries grow (Li, 2026; Kang et al., 2026). We therefore compare the selection under test with reference selections obtained from smaller pools.

Pool construction and execution. For a task x generated from a chain τ, the target pool is defined as $P_{\text{tar}} = \{s \mid s \in \tau\}$. SKILLDIVER first executes the agent R_{tar} times in the target pool to obtain reference selections:

$$Y_{\text{tar}}^{(r)} = \text{Sel}(A, x, P_{\text{tar}}), \qquad r = 1, \ldots, R_{\text{tar}}. \tag{1}$$

It then executes the agent once in the full pool P_{full}, which represents the realistic skill library including all skills available to the agent, to obtain the selection under test Y_{full}. If Y_{full} contains skills outside P_{tar}, SKILLDIVER forms the adaptive pool by adding these skills to the target pool:

$$P_{\text{adp}} = P_{\text{tar}} \cup \{s \in Y_{\text{full}} \mid s \notin P_{\text{tar}}\}. \tag{2}$$

The agent then executes the task R_{adp} times in the adaptive pool to obtain additional reference selections:

$$Y_{\text{adp}}^{(r)} = \text{Sel}(A, x, P_{\text{adp}}), \qquad r = 1, \ldots, R_{\text{adp}}. \tag{3}$$

If $Y_{\text{full}} \subseteq P_{\text{tar}}$, the adaptive-pool executions are skipped.

Selection comparison. Skill selection can vary across runs, so a single reference execution may miss a possible selection. Repeating the target-pool and adaptive-pool executions provides multiple opportunities to observe that selection. The differential module reports a deviation only if no reference execution reproduces the selection under test. Formally, the deviation is defined as:

$$\text{SelDev}(x) = \mathbb{I}\left[Y_{\text{full}} \notin \{Y_{\text{tar}}^{(r)}\}_{r=1}^{R_{\text{tar}}} \cup \{Y_{\text{adp}}^{(r)}\}_{r=1}^{R_{\text{adp}}}\right]. \tag{4}$$

Reproduction requires exact equality between skill sets, including the empty set when no skill is selected. A reference execution can also repeat an incorrect selection under test, causing it to go unreported. Section 4.2 evaluates this rule against benchmark labels and tests stricter matching thresholds. Algorithm 1 in Appendix F.1 gives the procedure.

3.4 FEEDBACK-GUIDED SCHEDULING

SKILLDIVER uses feedback from prior testing rounds to guide the selection of skill chains for subsequent task generation. The scheduling favors skills with fewer prior tests to improve coverage, while assigning higher weights to skills with higher observed deviation rates to prioritize scenarios that have exhibited selection deviations. After each testing round, SKILLDIVER updates the skill weights based on these two signals.

Skill weights. Before round t, let $n_t(s)$ denote the number of previous rounds in which skill s is included in the selected chain. Let $d_t(s)$ denote the number of those rounds that reported a selection deviation. After round t, $n_{t+1}(s)$ increases by one for every skill in the selected chain, and $d_{t+1}(s)$ increases by one for each such skill if the round reports a deviation. Thus, each deviation report is associated with the skills used to generate the corresponding task, since the test does not identify which skill is responsible for the deviation. We calculate the scheduling weight of skill s as

$$p_t(s) = (1 + e_t(s))/(1 + n_t(s)), \tag{5}$$

where $e_t(s)$ is the deviation rate among earlier tasks generated using skill s:

$$e_t(s) = \begin{cases} d_t(s)/n_t(s), & n_t(s) > 0, \\ 0, & n_t(s) = 0 \end{cases}. \tag{6}$$

The denominator reduces the weight of skills with more prior tests, encouraging coverage of less-tested skills, while the numerator increases it with the observed deviation rate, directing additional tests toward skills that exhibit more selection deviations. Using a rate rather than a raw deviation count prevents a skill from receiving a higher weight simply because it has more prior tests. Since $0 \leq e_t(s) \leq 1$, the deviation feedback can increase the weight by at most a factor of two at a fixed test count. The additive constants ensure that untested skills have an initial weight of 1 and that the weight is well-defined when $n_t(s) = 0$.

Chain sampling. Based on the skill weights, the weight of each chain $\tau \in \mathcal{T}$ is calculated as

$$w_t(\tau) = \sqrt{\sum_{s \in \tau} p_t(s)^2}, \tag{7}$$

and SKILLDIVER samples the next chain with probability

$$\Pr(\tau) = w_t(\tau)/(\sum_{\tau' \in \mathcal{T}} w_t(\tau')), \qquad \tau \in \mathcal{T}. \tag{8}$$

The Euclidean norm gives longer chains a higher base weight while reducing the effect of chain length compared with a direct sum. For a chain of k equally weighted skills, the weight grows as $\sqrt{k}$ rather than k. Thus, longer chains are generally more likely to be sampled, while a shorter chain containing substantially higher-weight skills can still receive a larger weight. The selected chain supplies the constraints for the next task, as described in Section 3.2.

4 EVALUATION

We evaluate whether SKILLDIVER discovers more skill selection deviations under a fixed test-case budget and whether its components meet their design objectives. We use Codex (OpenAI, 2025) with `GPT-5.5` (OpenAI, 2026) and OpenHands (Wang et al., 2025) with `GLM-5.3-Flash` (Z.ai, 2026) as the agents under test. For each task, the selection under test is the set of skills the agent selects from the full pool.

4.1 DEVIATION DISCOVERY

Setup. For deviation discovery, we construct four test settings with different full-pool sizes: one with all 199 SkillsBench (Li et al., 2026) skills and three with full pools of 50, 100, and 200 R3-Skill (Wang et al., 2026) skills. Each R3 full pool retains skills used to generate the task, with other candidates randomly sampled from the 2,050-skill library to reach the specified size. The baseline, GoS-Random-Direct, samples a skill or chain randomly from the same GoS candidate space and generates tasks directly from the complete skill documents. For both methods, candidate chains contain up to three skills (Xia et al., 2026). We allocate a test-case budget of 1,000 tasks to each method per agent and setting. Both methods use `GPT-5.4-mini` for generation. For each task, the agent runs once in the full pool and three times in each reference pool when invoked. We retain the default threshold of one match to limit false reports. Appendices C and F.5 provide execution and sampling details and the baseline prompt, respectively. We use three metrics. Selection Deviation Rate (SDR) is the percentage of tasks for which the differential module reports a deviation. Skill Coverage (SCov) counts distinct skills appearing in at least two reference selections for the same task across the target and adaptive pools. Similar to code coverage, SCov considers a skill covered only when it is selected by the agent, rather than used to generate a test case. SCov accumulates across tasks with different full pools, so it can exceed the size of an individual R3 full pool. Deviation-Associated Skills (DAS) counts distinct skills omitted or additionally selected across reported deviations, relative to the reference selections with the fewest differences from the selection under test. Appendix C.3 gives the complete calculation.

More selection deviations under the same test-case budget. SKILLDIVER achieves higher SDR than GoS-Random-Direct for both agents in all four settings (Table 1). Relative SDR gains range from 25.0% to 90.4% for Codex and from 9.8% to 39.3% for OpenHands. For example, on Codex R3-100, SKILLDIVER reports 158 deviations compared with 83 for the baseline. We further examine higher matching thresholds and find that using higher matching thresholds increases SKILLDIVER's SDR while preserving its higher SDR than GoS-Random-Direct (Appendix D.1).

The reports involve more distinct skills. DAS increases in all eight comparisons. For Codex on R3-100, DAS increases from 57 to 133. The tests also achieve higher SCov in every setting, with SCov increasing from 829 to 856 in the same Codex R3-100 setting. Coverage gains are smaller on SkillsBench, where both methods already cover nearly all 199 skills.

Human validation on generated tasks. To check whether reports on generated tasks reflect incorrect skill selections, we randomly sample 100 reported and 100 unreported cases from each of SKILLDIVER and GoS-Random-Direct, totaling 400 cases covering both agents and all four settings. Two annotators evaluate the selections without seeing the differential judgments. The two annotators label 77.0% and 79.5% of reported selections as incorrect, respectively, compared with 7.0% and 8.5% of unreported selections. The annotators give the same label to 358 of the 400 cases (89.5%, Cohen's $\kappa = 0.786$). We also use `GPT-5.5` as an LLM judge to assess these 400 cases (Section 4.2). Against the two sets of annotations, the differential module achieves F1 scores of 83.70%

Table 1: Selection deviation discovery results. Each pair reports SKILLDIVER/ GoS-Random-Direct. Gain is the relative improvement over the baseline. SDR is the percentage of tasks reported as deviations. SCov and DAS count distinct covered and deviation-associated skills across tasks.

Setting	Codex						OpenHands					
	SDR (%)		SCov		DAS		SDR (%)		SCov		DAS	
	Value	Gain	Value	Gain	Value	Gain	Value	Gain	Value	Gain	Value	Gain
SkillsBench	**18.7**/14.6	+28.1%	**199**/197	+1.0%	**97**/74	+31.1%	**16.3**/11.7	+39.3%	**199**/198	+0.5%	**93**/69	+34.8%
R3-50	**7.2**/5.6	+28.6%	**860**/797	+7.9%	**76**/48	+58.3%	**13.6**/12.2	+11.5%	**848**/773	+9.7%	**165**/95	+73.7%
R3-100	**15.8**/8.3	+90.4%	**856**/829	+3.3%	**133**/57	+133.3%	**19.1**/17.4	+9.8%	**872**/781	+11.7%	**179**/137	+30.7%
R3-200	**29.5**/23.6	+25.0%	**881**/839	+5.0%	**264**/196	+34.7%	**26.8**/21.2	+26.4%	**862**/801	+7.6%	**245**/182	+34.6%

Table 2: Judgment reliability on Codex's selections under test. Each pair reports differential module / LLM judge scores (%). Selections that differ from benchmark skill sets form the positive class.

Benchmark	Accuracy	Precision	Recall	F1
SkillsBench	**78.02**/49.45	**100.00**/94.74	**68.25**/28.57	**81.13**/43.90
SkillRet	**96.33**/63.56	**100.00**/98.48	**95.95**/60.75	**97.93**/75.14
R3-Skill	**80.00**/71.94	**100.00**/96.74	**61.50**/47.59	**76.16**/63.80
SRA-Bench	**86.07**/64.07	**99.19**/93.88	**71.51**/26.74	**83.11**/41.63
Average	**85.11**/62.26	**99.80**/95.96	**74.30**/40.91	**84.58**/56.12

and 84.57%, compared with 56.60% and 54.21% for the LLM judge, respectively. Appendix C.2 shows the sampling and annotation procedure, and Appendix D.2 gives the results.

Larger pools and longer skill chains have higher SDR. SDR increases with R3 full-pool size for both agents, consistent with prior observations of less reliable selection in larger libraries (Li, 2026; Zheng et al., 2026b; Kang et al., 2026). Tasks generated from longer skill chains also exhibit higher SDRs across full-pool sizes. For SKILLDIVER on R3-200, the SDRs aggregated across both agents are 22.7%, 44.0%, and 57.6% for skill chains containing one, two, and three skills (Appendix D.3).

Omissions dominate the reported cases. We pool the 2,616 deviation reports from SKILLDIVER and GoS-Random-Direct across both agents and all four settings. Relative to reference selections, omission-only deviations account for 78.5% of reported deviations. Addition-only deviations account for 11.4%, while deviations involving both omissions and additions account for 10.1% (Appendix D.4). One case illustrates a deviation involving both omissions and additions. The task asks the agent to trace trainer hierarchies and data flow in a library codebase. The full-pool execution selects `discover-important-function` and recognizes after reading it that the skill concerns fuzzing. Despite recognizing the mismatch, the agent continues without selecting the `trl` skill selected in the reference executions. Appendix E.5 details this case.

4.2 DIFFERENTIAL JUDGMENT RELIABILITY

Setup. To assess judgment reliability, we compare the differential module with an LLM judge on the same Codex selections under test. A selection under test is correct if it matches the benchmark skill set, and wrong otherwise. We use all 91 official SkillsBench (Li et al., 2026) tasks and sample 354 SkillRet (Kang et al., 2026), 360 R3-Skill (Wang et al., 2026), and 359 SRA-Bench (Su et al., 2026) tasks. For each task, we use its benchmark-labeled skills to construct the target pool because the original task-generation process is unavailable. The full pool contains all 199 SkillsBench skills or 200 skills for the other benchmarks, retaining the labeled skills and sampling the remaining candidates. Codex runs once in the full pool and three times in each reference pool (i.e., target pool and adaptive pool). By default, one matching reference selection suffices to withhold a deviation report. The LLM judge uses `GPT-5.5` to assess each selection under test. We report accuracy, precision, recall, and F1, treating wrong selections as the positive class. Appendices C and F.5 provide sampling details and the judge prompt, respectively.

The differential module achieves higher F1 than the LLM judge. Under this evaluation protocol, the differential module achieves an average F1 of 84.58%, exceeding the LLM judge by 28.46 percentage points (Table 2). It scores higher on every metric and benchmark, with the largest gains in recall and F1. One case illustrates how the differential module identifies an incorrect selection

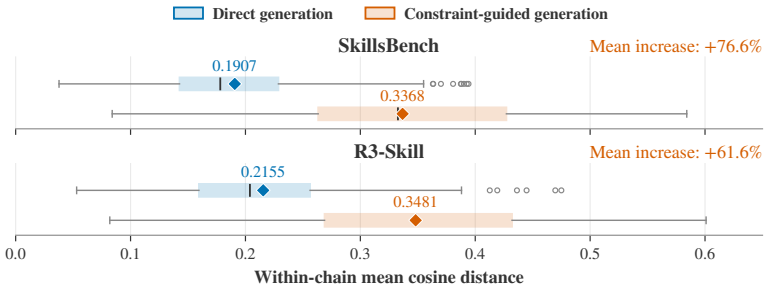

Figure 3: Task diversity with skill chains held fixed. Boxes show the interquartile range of per-chain scores. Vertical lines mark medians and diamonds mark means. Whiskers extend to the most extreme observations within 1.5 interquartile ranges of the box, with outliers shown as circles.

missed by the LLM judge. The task asks which postal code region has the most businesses, and the full-pool execution selects `Web Search`. The LLM judge accepts this selection. The reference executions instead select `Yelp aggregation` for the database query in every target-pool and adaptive-pool execution, even when `Web Search` is available. The differential module reports the selection under test because none of the reference executions reproduces it. Appendix E.6 presents the details for this case. Section 4.1 also compares the differential module and the LLM judge on 400 generated-task cases using human annotations as correctness labels.

False negatives dominate the judgment errors. We further analyze false negatives and false positives of the differential module. False negatives occur when a reference execution reproduces an incorrect selection under test. The agent can consistently omit the same labeled skill or choose the same broader alternative across pools, a pattern similar to common-mode failures. Variation between executions can also cause one reference execution to reproduce the incorrect selection, and one matching reference execution is enough to withhold a report even if the other executions select differently. The single false positive occurs when the selection under test matches the benchmark skill set but none of the reference executions reproduces it. Appendices D.5 and E.7 give per-benchmark error counts and individual cases.

Requiring more matching reference executions increases recall. To examine how the matching threshold affects judgment reliability, we compare thresholds requiring one, two, or three matching reference executions. We apply all three thresholds to the same recorded target-pool and adaptive-pool executions (Appendix D.6). Increasing the requirement from one to three matches raises average recall from 74.30% to 89.28% and F1 from 84.58% to 93.69%, while precision decreases slightly from 99.80% to 98.88%. Under a higher threshold, a single matching reference execution no longer suffices to suppress a deviation report, although a selection repeated across enough reference executions can still remain undetected.

4.3 Task Diversity and Naturalness

Setup. We compare constraint-guided generation with direct generation on the same skill chains to measure variation in tasks generated from the same skills. For each of SkillsBench and R3-Skill, both methods generate five tasks for each of the same 200 randomly sampled chains. For each chain, we compute the mean pairwise cosine distance among its task embeddings and average these scores across chains as an objective proxy metric to measure diversity and coverage. This metric requires no manual annotation, avoiding subjective bias. We assess task naturalness with `GPT-5.5` based on CheckEval (Lee et al., 2025). Appendix C.4 specifies the embedding model and calculation.

Constraint guidance produces more diverse tasks. The mean diversity score rises from 0.1907 with direct generation to 0.3368 with constraint guidance on SkillsBench, and from 0.2155 to 0.3481 on R3-Skill (Figure 3). These are relative increases of 76.6% and 61.6%. Constraint guidance achieves a higher diversity score on 173 of 200 SkillsBench chains and 167 of 200 R3-Skill chains. For example, all five tasks generated directly from `sympy` ask for equation solving, calculus, or algebraic manipulation. From the same document, SkillDiver extracts nine constraint groups corresponding to distinct documented uses. Across five generation runs, SkillDiver samples five of these groups and generates tasks involving physics calculations, algebraic simplification, Python

Table 3: Ablation results on Codex. Each entry reports SDR (%) / SCov / DAS.

Setting	Full method	w/o Constraint Guidance	w/o Feedback
SkillsBench	**18.7** / **199** / **97**	14.4 / **199** / 74	15.8 / 198 / 87
R3-50	**7.2** / 860 / **76**	6.0 / **882** / 52	6.4 / 737 / 55
R3-100	**15.8** / 856 / **133**	9.7 / **885** / 89	12.1 / 761 / 89
R3-200	**29.5** / 881 / **264**	25.2 / **893** / 245	26.2 / 795 / 215

code for evaluating formulas, LaTeX output, and geometry calculations. Table 14 compares the task distributions across these constraint groups.

Both methods receive high naturalness scores. Both methods score above 99.5% on both SkillsBench and R3-Skill. Appendix C.4 gives the checklist, scoring procedure, and results.

4.4 Effects of Constraint Guidance and Feedback

Setup. We use two ablations to separate the effects of task generation and skill scheduling on deviation discovery. Both use the Codex discovery settings and a budget of 1,000 tasks. The variant w/o Constraint Guidance generates directly from complete skill documents while using the full method's skill chains and full pools. The variant w/o Feedback randomly selects chains while retaining constraint-guided generation. We reuse the full method's results from Section 4.1.

Constraint guidance finds more deviations with similar skill coverage. Compared with w/o Constraint Guidance, SkillDiver achieves 17.1% to 62.9% higher SDR and 7.8% to 49.4% higher DAS across the four settings. SCov is unchanged on SkillsBench and 1.3% to 3.3% lower on R3. Together with the diversity results in Section 4.3, the higher SDR and DAS are consistent with broader task variation for the same skills being associated with more detected selection deviations.

Feedback expands the range of skills exercised. Compared with w/o Feedback, SkillDiver achieves higher values for all three metrics in every setting. SCov increases by 10.8% to 16.7% on R3, while SDR increases by 12.5% to 30.6% across the four settings. The coverage gain is consistent with the scheduler's preference for skills with fewer prior tests (Section 3.4).

5 Related Work

Skill selection and evaluation. Existing methods improve skill retrieval and routing using skill documents and relations between skills (Zheng et al., 2026b; Xiao et al., 2026; Liu et al., 2026). SkillRet, SRA-Bench, R3-Skill, and SkillsBench provide benchmarks with labeled tasks or reference skills for evaluating selection quality (Kang et al., 2026; Su et al., 2026; Wang et al., 2026; Li et al., 2026). Extending these benchmarks may require constructing tasks and validating skill associations, especially when alternative selections are possible. SkillDiver instead generates tasks from skill documents and assesses selections through reference executions without per-task correctness labels.

Automated testing of LLM agents. ToolEmu and RedCodeAgent construct or adapt test scenarios and evaluate agent responses, execution outcomes, or safety trajectories (Ruan et al., 2024; Guo et al., 2026). MetaSpace compares behavior across related instances through metamorphic testing (Xu et al., 2026). SkillDiver applies differential testing to skill selection by comparing selected skill sets for the same task across different skill pools, and uses extracted constraints and testing feedback to guide task generation and skill chain sampling. Appendix A extends this section to constraint-guided and feedback-guided test generation.

6 Conclusion

We present SkillDiver, an automated testing framework for evaluating agent skill selection. It combines constraint-guided task generation with adaptive differential testing to generate tasks from skill documents and assess selections without per-task correctness labels. Across two agents and four settings, SkillDiver discovers more selection deviations than GoS-Random-Direct. Constraint guidance increases task diversity, and the differential module achieves higher F1 than an LLM judge on four labeled benchmarks. See Appendix B for discussion of the criterion and evaluation scope.

Reproducibility Statement

Section 2 defines the testing objective and selection representation. Section 3 specifies the constraints, pool construction, differential rule, and scheduler. Appendix F.1 gives the judgment procedure in Algorithm 1. Section 4 gives the models, budgets, reference counts, and comparison conditions. Appendix C provides the execution, sampling, and scoring details, and Appendix F.2 reproduces the complete generation prompts and input formats. Appendix C.2 gives the human annotation procedure, and Appendix D.2 reports the results from both annotators. The evaluation pipeline records generated tasks, skill pools, model identifiers, and execution traces. We also provide an anonymous download link for the source code and experimental records in the supplementary material. Our code will be released upon acceptance.

AI Use Statement

Our method incorporates generative AI for constraint extraction and task generation, as described in Section 3. Codex is also used for code implementation. The generated code is validated using predefined test cases and manually inspected by the author. Additionally, we include LLM-based judging and task generation for baselines in our experiments, as described in Section 4. A generative AI assistant also supports literature search and discovery and language editing. All AI-assisted outputs are reviewed and verified by the authors, who remain responsible for the final content, implementation, and claims of this paper.

References

Syed Yusuf Ahmed, Shiwei Feng, Chanwoo Bae, Calix Barrus, and Xiangyu Zhang. Specops: A fully automated ai agent testing framework in real-world gui environments. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, ICSE '26, pp. 3091–3103. ACM, April 2026. doi: 10.1145/3744916.3787778. URL http://dx.doi.org/10.1145/3744916.3787778.

Anthropic. Equipping agents for the real world with Agent Skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025. Last accessed Aug. 2026.

Samuel Arcadinho, David Oliveira Aparicio, and Mariana S. C. Almeida. Automated test generation to evaluate tool-augmented LLMs as conversational AI agents. In Dieuwke Hupkes, Verna Dankers, Khuyagbaatar Batsuren, Amirhossein Kazemnejad, Christos Christodoulopoulos, Mario Giulianelli, and Ryan Cotterell (eds.), *Proceedings of the 2nd GenBench Workshop on Generalisation (Benchmarking) in NLP*, pp. 54–68, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.genbench-1.4. URL https://aclanthology.org/2024.genbench-1.4/.

Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W. Tsang. Skilldag: Self-evolving typed skill graphs for llm skill selection at scale, 2026. URL https://arxiv.org/abs/2606.03056.

Jianming Chen, Yawen Wang, Junjie Wang, Zhe Liu, Qing Wang, and Fanjiang Xu. Orchjail: Jailbreaking tool-calling text-to-image agents by orchestration-guided fuzzing. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=VviX6pPCgV.

Yanping Chen, Weijie Shi, Wen Yang, and Jiajie Xu. Task decomposition-guided reranking for adaptive agent skill retrieval, 2026b. URL https://arxiv.org/abs/2607.06283.

Yu Chen, Ruishuo Chen, Xun Wang, Zhuoran Li, and Longbo Huang. Optimal skill selection for llm agents with provable bicriteria guarantees, 2026c. URL https://arxiv.org/abs/2608.19993.

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models, 2023. URL https://arxiv.org/abs/2212.14834.

Lihui Ding, Zihan Guo, Bingwei Lu, Chenyu Zhou, Yuanjian Zhou, Weinan Zhang, Jianghao Lin, and Dongdong Ge. Skill2query: Exploiting skill structure to generate pseudo-queries for agent skill retrieval, 2026. URL https://arxiv.org/abs/2608.16071.

Khashayar Etemadi, Bardia Mohammadi, Zhendong Su, and Martin Monperrus. Mokav: Execution-driven differential testing with llms. *J. Syst. Softw.*, 230(C), December 2025. ISSN 0164-1212. doi: 10.1016/j.jss.2025.112571. URL https://doi.org/10.1016/j.jss.2025.112571.

Xueping Gao. Compositional skill routing for llm agents: Decompose, retrieve, and compose. *arXiv preprint arXiv:2606.18051*, 2026.

Patrice Godefroid, Michael Y. Levin, and David Molnar. Sage: whitebox fuzzing for security testing. *Commun. ACM*, 55(3):40–44, March 2012. ISSN 0001-0782. doi: 10.1145/2093548.2093564. URL https://doi.org/10.1145/2093548.2093564.

Chengquan Guo, Chulin Xie, Yu Yang, Zhaorun Chen, Zinan Lin, Xander Davies, Yarin Gal, Dawn Song, and Bo Li. Redcodeagent: Automatic red-teaming agent against diverse code agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IyIaAOihmZ.

Zhenlan Ji, Daoyuan Wu, Pingchuan Ma, Zongjie Li, and Shuai Wang. Testing and understanding erroneous planning in llm agents through synthesized user inputs, 2024. URL https://arxiv.org/abs/2404.17833.

Donghong Jiang, Endian Lin, Luoping Cui, Hanqing Liu, Mingjie Liu, Fan Yang, Hong Wang, Zhao Yang, and Chuang Zhu. Skillreason: Reasoning-enhanced agent skill retrieval for implicit user requests, 2026. URL https://arxiv.org/abs/2608.08640.

Ryangkyung Kang, Hongcheol Cho, and Youngeun Kim. Skillret: A large-scale benchmark for skill retrieval in llm agents, 2026. URL https://arxiv.org/abs/2605.05726.

Yukyung Lee, JoongHoon Kim, Jaehee Kim, Hyowon Cho, Jaewook Kang, Pilsung Kang, and Najoung Kim. CheckEval: A reliable LLM-as-a-judge framework for evaluating text generation using checklists. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 15771–15798, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.796. URL https://aclanthology.org/2025.emnlp-main.796/.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Xiaoxiao Li. When single-agent with skills replace multi-agent systems and when they fail, 2026. URL https://arxiv.org/abs/2601.04748.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills, 2026. URL https://arxiv.org/abs/2604.05333.

Fengyu Liu, Yuan Zhang, Jiaqi Luo, Jiarun Dai, Tian Chen, Letian Yuan, Zhengmin Yu, Youkun Shi, Ke Li, Chengyuan Zhou, Hao Chen, and Min Yang. Make agent defeat agent: automatic detection of taint-style vulnerabilities in llm-based agents. In *Proceedings of the 34th USENIX Conference on Security Symposium*, SEC '25, USA, 2025. USENIX Association. ISBN 978-1-939133-52-6.

Jiaqi Luo, Jiarun Dai, Fengyu Liu, Songyang Peng, Youkun Shi, Tong Bu, Geng Hong, Xudong Pan, and Yuan Zhang. Autonomy comes with costs: Detecting Denial-of-Service vulnerabilities caused by resource abusing in LLM-based agents. In *35th USENIX Security Symposium (USENIX Security 26)*, pp. 3991–4010, Baltimore, MD, August 2026. USENIX Association. ISBN 978-1-939133-58-8. URL https://www.usenix.org/conference/usenixsecurity26/presentation/luo.

Jianan Ma, Xiaohu Du, Ruixiao Lin, Yaoxiang Bian, Jialuo Chen, Yunhao Feng, Xiaofang Yang, Shiwen Cui, Changhua Meng, Xinhao Deng, Jingyi Wang, and Zhen Wang. Aseval: Automated trajectory-level security testing for autonomous agents, 2026. URL https://arxiv.org/abs/2605.22321.

William M McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998.

Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. Large language model guided protocol fuzzing. In *NDSS*, 2024.

OpenAI. Codex. https://openai.com/codex/, 2025. Last accessed Aug. 2026.

OpenAI. Introducing GPT-5.5. https://openai.com/index/introducing-gpt-5-5/, 2026. Last accessed Aug. 2026.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox, 2024. URL https://arxiv.org/abs/2309.15817.

Dohyun Ryu, Giyeol Kim, Daeun Lee, Seongjin Kim, Seungjin Bae, Junghwan Rhee, and Taegyu Kim. Differential fuzzing for data distribution service programs with dynamic configuration. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 807–818, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3695073. URL https://doi.org/10.1145/3691620.3695073.

Dongdong She, Adam Storek, Yuchong Xie, Seoyoung Kweon, Prashast Srivastava, and Suman Jana. Fox: Coverage-guided fuzzing as online stochastic control, 2024. URL https://arxiv.org/abs/2406.04517.

Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Driller: Augmenting fuzzing through selective symbolic execution. In *NDSS*, volume 16, pp. 1–16, 2016.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic ai, 2026. URL https://arxiv.org/abs/2604.24594.

Boyin Tan, Xiaowei Huang, and Youcheng Sun. Skill coverage: A test adequacy metric for agent skills, 2026. URL https://arxiv.org/abs/2606.20659.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025.

Zifei Wang, Wei Wen, Qiang Ji, Keyu Chen, Ruizhi Qiao, and Xing Sun. Skill is not document: Query-conditioned compatibility for llm agent skill routing, 2026. URL https://arxiv.org/abs/2606.03565.

Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639121. URL https://doi.org/10.1145/3597503.3639121.

Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. Grasp: Graph-structured skill compositions for llm agents, 2026. URL https://arxiv.org/abs/2604.17870.

Jinying Xiao, Bin Ji, Shasha Li, Xiaodong Liu, Ma Jun, Jiacheng Jie, Chao Wang, Nyima Tashi, and Jie Yu. Skillsight: Seeing through shared descriptions for accurate skill retrieval. *arXiv preprint arXiv:2607.18785*, 2026.

Gengyang Xu, Dongwei Xiao, Yiteng Peng, and Shuai Wang. Metaspace: Metamorphic testing for spatial cognition in embodied agents. *Proc. ACM Program. Lang.*, 10(OOPSLA1), April 2026. doi: 10.1145/3798212. URL https://doi.org/10.1145/3798212.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. In *First Workshop on Agent Skills*, 2026. URL https://openreview.net/forum?id=Er0p92BsmW.

Chenyuan Yang, Yinlin Deng, Runyu Lu, Jiayi Yao, Jiawei Liu, Reyhaneh Jabbarvand, and Lingming Zhang. Whitefox: White-box compiler fuzzing empowered by large language models. *Proc. ACM Program. Lang.*, 8(OOPSLA2), October 2024. doi: 10.1145/3689736. URL https://doi.org/10.1145/3689736.

Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang, and Taesoo Kim. QSYM : A practical concolic execution engine tailored for hybrid fuzzing. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 745–761, Baltimore, MD, August 2018. USENIX Association. ISBN 978-1-939133-04-5. URL https://www.usenix.org/conference/usenixsecurity18/presentation/yun.

Z.ai. GLM-5.3-Flash: Frontier Intelligence, Flash Cost. https://z.ai/blog/glm-5.3-flash, 2026. Last accessed Aug. 2026.

Michał Zalewski. american fuzzy lop. https://lcamtuf.coredump.cx/afl/, 2014. Last accessed Aug. 2026.

Yukai Zhao, Menghan Wu, Xing Hu, and Xin Xia. Hfuzzer: Testing large language models for package hallucinations via phrase-based fuzzing. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 2746–2758. IEEE Press, 2025. doi: 10.1109/ASE63991.2025.00225. URL https://doi.org/10.1109/ASE63991.2025.00225.

Jingyuan Zheng, Dongjing Wang, Xin Zhang, Hao Chen, Youhuizi Li, Xudong Shen, Haiping Zhang, Butian Huang, Dongjin Yu, and Guandong Xu. Skillselect-serve: Qos-aware budgeted skill service recommendation for llm agents, 2026a. URL https://arxiv.org/abs/2607.00011.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. Skillrouter: Skill routing for llm agents at scale, 2026b. URL https://arxiv.org/abs/2603.22455.

Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications, 2026. URL https://arxiv.org/abs/2605.07358.

APPENDIX CONTENTS

A Extended Related Work **16**

A.1 Skill Selection and Routing 16

A.2 Evaluating Skill Selection 16

A.3 Automated Testing of LLM Agents 16

A.4 Constraint-Guided and Feedback-Guided Test Generation 16

B Discussion **17**

B.1 Reference Matching 17

B.2 Evaluation Scope 17

C Experimental Details **18**

C.1 Execution and Pool Sampling 18

C.2 Human Validation Protocol 18

C.3 Discovery Metrics 19

C.4 Task Diversity and Naturalness Scoring 19

C.5 Ablation Setting 19

D Additional Results and Analysis **20**

D.1 Deviation Discovery at Higher Matching Thresholds 20

D.2 Human Validation on Generated Tasks 20

D.3 Full-Pool Size and Skill Chain Length 21

D.4 Types of Selection Deviations 21

D.5 Judgment Error Analysis 22

D.6 Effect of the Matching Threshold 23

E Case Studies **23**

E.1 Task Generation for Different Skill Uses 23

E.2 Agreement Between Target and Full Pools 24

E.3 Alternative Selection for Audio Conversion 25

E.4 Reported Deviation for NPZ Validation 25

E.5 Examples of Selection Deviation Types 26

E.6 A Selection Error Missed by the LLM Judge 28

E.7 Judgment Error Cases 29

F Method Details and Evaluation Prompts **30**

F.1 Adaptive Differential Testing 30

F.2 Prompts of SKILLDIVER 30

F.3 Constraint Extraction Example 34

F.4 Task Generation Example 37

F.5 Baseline and Evaluator Prompts 38

A Extended Related Work

A.1 Skill Selection and Routing

Existing selection methods improve skill retrieval and adapt choices as a task progresses. SkillRouter uses skill documents for retrieval and reranking, while SkillSight reduces interference from shared descriptions (Zheng et al., 2026b; Xiao et al., 2026). Other approaches improve retrieval through reasoning supervision for implicit requests or queries generated from skill structure (Jiang et al., 2026; Ding et al., 2026). Graph of Skills and SkillDAG model dependencies and other relations between skills, while SkillReranker updates candidate rankings during task execution (Liu et al., 2026; Bai et al., 2026; Chen et al., 2026b). SkillSelect-Serve incorporates token budgets, risk, and tool availability into selection (Zheng et al., 2026a). Chen et al. model skill selection as an optimization problem and propose the Best Prefix Selection algorithm (Chen et al., 2026c). These methods aim to improve skill selection. SkillDiver tests the agent's selections using tasks generated from skill chains constructed with GoS.

A.2 Evaluating Skill Selection

SkillRet, SRA-Bench, R3-Skill, and SkillsBench provide labeled tasks or curated reference skills for evaluating selection quality (Kang et al., 2026; Su et al., 2026; Wang et al., 2026; Li et al., 2026). We use these benchmarks to assess the reliability of differential judgment. Extending such evaluation to new skills or scenarios requires constructing tasks and validating their skill associations. The skills used to generate a task do not necessarily define its only valid selection, so validating each task label requires checking plausible alternatives and increases annotation cost. SkillDiver generates tasks from skill documents and uses reference executions to assess selections without requiring per-task correctness labels.

A.3 Automated Testing of LLM Agents

Automated agent testing covers planning, tool use, conversation, and behavior in executable environments (Ji et al., 2024; Arcadinho et al., 2024; Ahmed et al., 2026). ToolEmu and RedCodeAgent construct or adapt test scenarios and evaluate agent responses, execution outcomes, or safety trajectories (Ruan et al., 2024; Guo et al., 2026). ASEval also uses actions in execution trajectories for security evaluation, while other fuzzing methods manipulate inputs or environments to expose agent vulnerabilities (Ma et al., 2026; Liu et al., 2025; Luo et al., 2026). MetaSpace compares behavior across related instances through metamorphic testing (Xu et al., 2026). SkillDiver applies differential testing to skill selection by holding each task fixed, varying the available skill pool, and comparing the selected skill sets.

A.4 Constraint-Guided and Feedback-Guided Test Generation

AFL uses coverage feedback to guide fuzzing, and Fox formulates coverage-guided fuzzing as online stochastic control (Zalewski, 2014; She et al., 2024). SAGE derives path constraints through symbolic execution, while Driller and QSYM combine mutation with symbolic execution to reach paths that mutation alone may miss (Godefroid et al., 2012; Stephens et al., 2016; Yun et al., 2018). Differential fuzzing is also used to test data distribution services with dynamic configurations (Ryu et al., 2024). LLMs is also used to generate programs, compiler optimization tests, and stateful protocol messages for fuzzing (Xia et al., 2024; Yang et al., 2024; Meng et al., 2024; Deng et al., 2023). Recently, some research has also applied fuzzing to LLM issues such as package hallucination and jailbreaking (Zhao et al., 2025; Chen et al., 2026a). SkillDiver extracts constraints from skill documents to generate tasks for different skill uses. Test counts and observed selection deviations guide which skill chains are sampled for subsequent task generation.

B Discussion

B.1 Reference Matching

Reference matching addresses the ambiguity of skill selection for generated tasks. The source skills provide the basis for constructing a request, but other skills or the agent's existing capabilities may also support it. SkillDiver assesses the selection under test by checking whether it recurs when fewer candidate skills are available, motivated by observations that selection becomes less reliable as the candidate pool grows (Li, 2026; Bai et al., 2026). The adaptive pool retains the skills in the selection under test alongside the source skills, allowing an alternative selection to recur even when it contains skills outside the target pool.

A selection deviation reflects a change in the agent's selection behavior under a changed candidate pool, and its interpretation depends on how that behavior varies across executions. An incorrect selection may be reproduced in a reference execution, causing the differential module to withhold a report. Conversely, a valid selection may fail to recur because multiple skill sets can support the task or the agent may vary its choices across executions. Thus, failure to reproduce the selection provides a signal of a potential error, but does not establish that the selection is incorrect. We then evaluate whether this signal agrees with external correctness judgments. Across four labeled benchmarks, the selection-deviation signal achieves an average precision of 99.80% and recall of 74.30% against the benchmark labels (Section 4.2). Human validation provides complementary evidence: the two annotators judge 77.0% and 79.5% of reported selections incorrect, compared with 7.0% and 8.5% of unreported selections, respectively (Appendix D.2). These results indicate that selection deviations are strongly associated with incorrect selections in the evaluated settings.

The matching threshold determines how much recurrence is required to withhold a report. Our discovery experiments use three executions per reference pool when invoked and withhold a report when at least one reference selection matches the selection under test. We use this threshold to favor precision while allowing alternative selections to recur across the repeated reference executions. Requiring more matches increases recall and F1 while reducing precision on the recorded benchmark executions (Appendix D.6). With a higher threshold, a single reference execution that repeats an incorrect selection no longer suffices to suppress a report, although an incorrect selection that recurs across enough reference executions can remain undetected.

B.2 Evaluation Scope

The benchmark evaluation provides labeled selections for analyzing false positives and false negatives under a fixed correctness criterion. It constructs target pools from benchmark-labeled skills because the original task-generation process is unavailable. This construction may favor reference selections consistent with the labels used to evaluate the judgments. Human validation complements this evaluation by examining generated tasks under the discovery protocol, where target pools are constructed from the source skills and human labels are used only for evaluation. We compare the differential module and the LLM judge on the same 400 generated-task cases from SkillDiver and GoS-Random-Direct, sampled from reported and unreported selections for each method. The higher F1 against each annotator's labels supports the usefulness of the differential signal on these generated tasks, beyond its agreement with benchmark skill sets (Appendix D.2).

Task quality should also be interpreted in relation to the selection objective. The naturalness assessment measures task phrasing and contextual coherence, whereas successful execution additionally requires the inputs and environment necessary to complete the task. A naturally phrased request may therefore still omit information needed for execution. The differential evaluation compares the same task across the full, target, and adaptive pools, holding the task request fixed while varying the available skills. Task incompleteness may reduce the executability of a generated task, but may not affect our assessment of skill-selection behavior when the agent responds similarly across the evaluated pools.

The discovery experiments cover two agent-model combinations and four settings derived from SkillsBench and R3-Skill; the benchmark evaluation uses Codex across four datasets. Although R3-Skill contains 2,050 skills, each evaluated full pool contains at most 200, with sampling used to keep the available skills within the agents' context limits. The evaluation results therefore characterize selection behavior under the evaluated agents, models, skill libraries, task settings, and pool

sizes. Whether the same behavior and differential signal hold for substantially different agents, skill libraries, or larger candidate pools remains to be established.

Repeated reference executions provide multiple observations for each task, but do not characterize variation across independent runs of the complete testing pipeline. Generated tasks and agent selections may also vary across runs, and changes to models or execution environments may affect these outcomes over time. We record the generated tasks, skill pools, model identifiers, configurations, and execution traces to support inspection and reproduction of the reported results.

C Experimental Details

This appendix gives the execution, sampling, and scoring protocols for Section 4.

C.1 Execution and Pool Sampling

We run agents through Harbor and extract selected skills from the execution traces using the same matching rules for all methods. All experiments run on a Linux x86_64 server with two Intel Xeon Gold 6226R processors (64 logical CPUs) and 157 GiB of memory. Codex uses the model's default temperature. Constraint extraction uses `GPT-5.4-mini` at temperature 0.0, and task generation uses the same model at temperature 0.7. The tested agent models and reference execution counts are given in Section 4.1.

For R3, each task's full pool retains the skills used by task generation and selects additional skills from the remaining library until it reaches 50, 100, or 200 skills. Loading all 2,050 skills can exceed the tested agent's context limit. The target and adaptive pools follow Section 3.3. Consequently, a setting such as R3-50 contains tasks with different 50-skill full pools, and aggregate skill coverage can exceed 50.

Reliability evaluation uses all 91 official SkillsBench tasks. For SkillRet, R3-Skill, and SRA-Bench, we independently sample 354, 360, and 359 tasks, respectively. We calculate the sample size based on the number of tasks in each benchmark, using a 95% confidence level and a 5% margin of error. The benchmark-labeled skills form the target pool, and the full pool retains these skills while selecting additional candidates from the remaining library. The positive class consists of selections under test that differ from the benchmark skill set.

C.2 Human Validation Protocol

Sampling. We randomly select cases from SkillDiver and GoS-Random-Direct in the discovery experiments of Section 4.1, which cover eight configurations: Codex and OpenHands on SkillsBench and the three R3 settings. For each method, we pool the results across these configurations and classify the cases by whether the differential module reports a skill selection deviation. We randomly select 100 cases from each of the reported and unreported classifications per method, then combine and shuffle each method's cases, giving 400 cases in total. Each case includes a generated task, the selection under test, visible skill catalog, and candidate skill documents. The final case set contains 203 Codex cases and 197 OpenHands cases, with the counts for each method and configuration shown in Table 4.

Table 4: Distribution of the 200 human validation cases per method across agents and settings.

	SkillDiver		GoS-Random-Direct	
Setting	Codex	OpenHands	Codex	OpenHands
SkillsBench	25	27	26	22
R3-50	18	24	33	21
R3-100	22	19	24	21
R3-200	31	34	24	29
Total	**96**	**104**	**107**	**93**

Annotation procedure. Two annotators assess whether each selection under test is correct for the task, using the same criteria. A `yes` label means that the selection is correct, and `no` means that it is incorrect. The annotators do not see the differential results or reference selections. We retain both sets of labels for all sampled cases.

C.3 Discovery Metrics

SDR is the number of reported deviation cases divided by the total number of test cases, expressed as a percentage. SCov counts the number of unique skills covered by the test cases. Due to model stochasticity, a skill is considered covered only if the agent selects it in at least two reference executions of the same task. We combine the three target-pool executions with the three adaptive-pool executions when available and include skills outside the chain used to generate the task. Repeated accesses within one execution count once, and SCov counts each covered skill once across all tasks. DAS counts distinct skills associated with reported selection deviations, providing candidates for improving skill descriptions or agent selection mechanisms. For each deviation case, we compare the selection under test with every reference selection and retain those with the fewest differing skills. A skill present only in the selection under test is considered additionally selected, while a skill present only in a retained reference selection is considered omitted. If multiple reference selections tie, we collect the omitted and additional skills from all of them. Each skill is counted once across all reported cases.

C.4 Task Diversity and Naturalness Scoring

For each dataset, both methods generate five tasks for each of the same 200 randomly sampled chains, giving 1,000 tasks per method. We embed each task with `text-embedding-3-small` and compute the ten pairwise cosine distances among the five tasks generated for each chain. We average these distances within each chain and then across chains to obtain the diversity score. Shared chains keep the source skills fixed when comparing generation methods. Figure 3 summarizes the distributions of per-chain scores and marks the means across chains.

The naturalness evaluator uses the five seed questions from CheckEval (Lee et al., 2025), covering repetition, contextual relevance, clarity, consistency of tone, and grammatical correctness. GPT-5.5 answers all five questions in one call per task. Figure 9 gives the complete evaluation prompt, including the five questions. Each task's score is the proportion of Yes answers. We average the task scores within each chain, then across the 200 chains.

Table 5: Naturalness scores on the tasks used for the diversity comparison. Scores are the percentage of positive CheckEval answers, averaged within and then across the same 200 skill chains per dataset.

Dataset	SkillDiver	Direct generation
SkillsBench	99.56	99.90
R3-Skill	99.70	99.68

Table 5 gives the complete scores. The slightly lower SkillsBench score for SkillDiver mainly comes from incomplete phrases or stray punctuation left by the rules that remove skill names.

C.5 Ablation Setting

The variant w/o Constraint Guidance reuses the full method's sequence of skill chains and full pools, but generates directly from complete skill documents. The variant w/o Feedback samples chains randomly and retains constraint-guided generation. Both use the Codex discovery configuration and 1,000 test cases per setting. The full method's discovery results are reused for comparison.

Constraint guidance leaves SCov unchanged on SkillsBench and reduces it by 1.3% to 3.3% on R3, despite increasing SDR and DAS. Fixing the skill chains and full pools does not fix which skills the agent selects for the generated tasks. SCov counts each covered skill once, so additional task scenarios for that skill do not increase its coverage count.

D Additional Results and Analysis

D.1 Deviation Discovery at Higher Matching Thresholds

We apply matching thresholds $k = 2$ and $k = 3$ to the discovery tasks and executions used in Section 4.1. A deviation is reported when fewer than k reference executions reproduce the selection under test. Note that we only check the results in Section 4.1 using the new matching thresholds and do not rerun the experiments themselves.

Table 6: SDR at matching thresholds $k = 2$ and $k = 3$ on the recorded discovery executions. Each pair reports SkillDiver/ GoS-Random-Direct. Gain is the relative improvement over the baseline.

Setting	$k = 2$				$k = 3$			
	Codex		OpenHands		Codex		OpenHands	
	SDR (%)	Gain	SDR (%)	Gain	SDR (%)	Gain	SDR (%)	Gain
SkillsBench	**21.3**/16.9	+26.0%	**20.0**/16.7	+19.8%	**24.1**/19.8	+21.7%	**23.9**/22.4	+6.7%
R3-50	**9.9**/9.1	+8.8%	**15.5**/15.4	+0.6%	**12.9**/12.6	+2.4%	**18.2**/18.0	+1.1%
R3-100	**18.7**/11.9	+57.1%	**21.6**/20.1	+7.5%	**21.9**/16.5	+32.7%	**24.2**/23.1	+4.8%
R3-200	**33.0**/27.3	+20.9%	**29.2**/23.7	+23.2%	**35.6**/31.5	+13.0%	**31.7**/26.2	+21.0%

Across SkillDiver's 8,000 tasks, SDR increases from 18.38% at $k = 1$ to 21.15% at $k = 2$ and 24.06% at $k = 3$. SkillDiver still achieves higher SDR than GoS-Random-Direct for both agents in all four settings at both thresholds (Table 6). The differences narrow in several settings: at $k = 3$, SDR on R3-50 is 12.9% versus 12.6% for Codex and 18.2% versus 18.0% for OpenHands.

D.2 Human Validation on Generated Tasks

Table 7: Human judgments for 100 reported and 100 unreported cases per method, each assessed by two annotators. Counts indicate whether the selection under test is correct (`yes`) or incorrect (`no`).

		SkillDiver		GoS-Random-Direct	
Case group	Human judgment	Annotator 1	Annotator 2	Annotator 1	Annotator 2
Reported	Correct (`yes`)	17	20	29	21
	Incorrect (`no`)	83	80	71	79
Unreported	Correct (`yes`)	89	86	97	97
	Incorrect (`no`)	11	14	3	3

Table 7 reports the human judgments collected using the sampling and annotation procedure in Appendix C.2. The two annotators give the same label to 358 of 400 cases (89.5% agreement, Cohen's $\kappa = 0.786$). Of these, 207 selections are judged correct by both annotators and 151 incorrect by both. Thirty-five disagreements occur among reported cases and seven among unreported cases. Among reported cases, the two annotators judge 77.0% and 79.5% of selections incorrect, respectively, corresponding to report precision for each annotator. Among unreported cases, 7.0% and 8.5% of selections are judged incorrect, respectively. Both annotators support most deviation reports and identify selection errors that the differential module misses.

The sample contains 200 cases from each generation method. For SkillDiver, the annotators judge 83% and 80% of reported selections incorrect, compared with 11% and 14% of unreported selections. For GoS-Random-Direct, the corresponding proportions are 71% and 79% for reported selections and 3% for unreported selections for both annotators.

The same 400 cases are also assessed using the LLM judge described in Section 4.2. Table 8 evaluates the judgments against the human annotations from each annotator, using the annotator-specific annotations as ground-truth labels and incorrect selections as the positive class. On these 400 cases, the differential module achieves higher F1 across both annotators, with scores of 83.70% and 84.57%, compared with 56.60% and 54.21% for the LLM judge, respectively. It also achieves

higher recall for both annotators, correctly identifying 91.67% and 90.34% of the selections labeled incorrect by the annotators.

Table 8: Judgment metrics (%) against each annotator's labels on 200 cases per task source and all 400 cases combined. Human annotations serve as ground-truth labels, with incorrect selections as the positive class.

Task source	Annotator	Judgment method	Accuracy	Precision	Recall	F1
SKILLDIVER	1	Differential module	86.00	83.00	88.30	85.57
	1	LLM judge	72.00	82.76	51.06	63.16
	2	Differential module	83.00	80.00	85.11	82.47
	2	LLM judge	70.00	79.31	48.94	60.53
GoS-Random-Direct	1	Differential module	84.00	71.00	95.95	81.61
	1	LLM judge	70.50	69.23	36.49	47.79
	2	Differential module	88.00	79.00	96.34	86.81
	2	LLM judge	67.50	71.79	34.15	46.28
Combined	1	Differential module	85.00	77.00	91.67	83.70
	1	LLM judge	71.25	77.32	44.64	56.60
	2	Differential module	85.50	79.50	90.34	84.57
	2	LLM judge	68.75	76.29	42.05	54.21

D.3 FULL-POOL SIZE AND SKILL CHAIN LENGTH

Table 9: SDR by skill chain length for SKILLDIVER. Each entry gives reports/tasks (SDR in percent). Each agent and setting contributes 1,000 tasks.

Setting	Agent	One skill	Two skills	Three skills
SkillsBench	Codex	183/988 (18.5)	4/12 (33.3)	—
SkillsBench	OpenHands	155/987 (15.7)	8/13 (61.5)	—
R3-50	Codex	41/830 (4.9)	9/64 (14.1)	22/106 (20.8)
R3-50	OpenHands	93/830 (11.2)	8/53 (15.1)	35/117 (29.9)
R3-100	Codex	103/823 (12.5)	15/61 (24.6)	40/116 (34.5)
R3-100	OpenHands	128/825 (15.5)	18/54 (33.3)	45/121 (37.2)
R3-200	Codex	187/818 (22.9)	25/57 (43.9)	83/125 (66.4)
R3-200	OpenHands	184/819 (22.5)	30/68 (44.1)	54/113 (47.8)

Table 9 shows higher SDR for longer skill chains across SkillsBench and the three R3 settings for both agents. For example, on SkillsBench, SDR rises from 18.5% to 33.3% for Codex and from 15.7% to 61.5% for OpenHands as chain length increases from one to two skills. These results show that tasks involving several operations may pose additional challenges for skill selection. Additionally, we find that large full pools also have higher SDR within each skill chain length for both agents on R3. For tasks generated from one skill, SDR rises from 4.9% to 22.9% for Codex and from 11.2% to 22.5% for OpenHands as the full pool grows from 50 to 200 skills. This aligns with the findings of other existing studies, suggesting that a vast pool of candidates can also make skill selection difficult.

D.4 TYPES OF SELECTION DEVIATIONS

We classify the 2,616 reports in Section 4.1 using the reference selections with the fewest differences from the selection under test, as in the DAS calculation. If several reference selections tie, we collect missing skills and additional skills separately across all tied reference selections. A report is an omission when it has only missing skills, an addition when it has only extra skills, and mixed when it has both. For both types of agents, missing skills is the main cause of reported deviations, accounting for 78.5% of the total (Table 10). The cluster communication case illustrates an omission of `erlang-distribution`, which provides guidance for the requested communication between remote processes (Appendix E.5). Additions and mixed deviations account for 11.4% and 10.1% of reports, respectively. We provide corresponding examples in Appendix E.5.3 and Appendix E.5.1.

Table 10: Types of reported deviations pooled across SKILLDIVER, GoS-Random-Direct, and all four discovery settings. Each type gives the number of reports and its percentage within the row. Omission means only missing skills, addition means only extra skills, and mixed means both, relative to the closest reference selections.

Agent	Reports	Omission	Addition	Mixed
Codex	1233	964 (78.2)	119 (9.7)	150 (12.2)
OpenHands	1383	1089 (78.7)	180 (13.0)	114 (8.2)
Total	2616	2053 (78.5)	299 (11.4)	264 (10.1)

D.5 JUDGMENT ERROR ANALYSIS

Table 11: Judgment results on Codex's selections under test. Each pair reports differential module / LLM judge counts. Incorrect selections are the positive class.

Benchmark	Tasks	TP	TN	FP	FN
SkillsBench	91	43 / 18	28 / 27	0 / 1	20 / 45
SkillRet	354	308 / 195	33 / 30	0 / 3	13 / 126
R3-Skill	360	115 / 89	173 / 170	0 / 3	72 / 98
SRA-Bench	359	123 / 46	186 / 184	1 / 3	49 / 126
Total	**1164**	**589 / 348**	**420 / 411**	**1 / 10**	**154 / 395**

Table 11 reports the details for the differential module and the LLM judge on the same selections under test from 1,164 labeled tasks. The differential module achieves higher accuracy, precision, recall, and F1 across all four benchmarks (Table 2). It produces 154 false negatives and one false positive, compared with 395 and 10 for the LLM judge, respectively. The reduction in false negatives accounts for most of the overall gain. The largest reduction occurs on SkillRet, where false negatives decrease from 126 to 13, followed by SRA-Bench, where they decrease from 126 to 49. On R3-Skill, false negatives decrease from 98 to 72, and the differential module has its lowest recall among the four benchmarks at 61.50%. False positives also decrease on every benchmark, so the differential module identifies more incorrect selections while incorrectly flagging fewer correct selections.

Table 12: Paired comparison of differential module and LLM judge judgments, showing shared and method-specific errors against benchmark labels.

Judgment correctness		Number of tasks				
Differential module	LLM judge	SkillsBench	SkillRet	R3-Skill	SRA-Bench	Total
Correct	Correct	41	224	225	227	717
Correct	Incorrect	30	117	63	82	292
Incorrect	Correct	4	1	34	3	42
Incorrect	Incorrect	16	12	38	47	113
Total		91	354	360	359	1164

Table 12 pairs the judgments by task to distinguish errors shared by the two methods from those unique to either method. The differential module is correct on 292 cases where the LLM judge is wrong, comprising 282 false negatives and 10 false positives by the LLM judge. Conversely, the LLM judge is correct on 42 cases where the differential module is wrong, comprising 41 false negatives and one false positive by the differential module. These results reveal complementary error patterns between the two methods, indicating that combining their judgments or signals may improve judgment reliability, although we leave such a combined criterion to future work. Appendix E.6 presents a case where only the LLM judge is wrong, and Appendix E.7 examines errors made by the differential module.

D.6 Effect of the Matching Threshold

To examine how the matching requirement affects judgment reliability, we compare the default threshold $k = 1$ with $k = 2$ and $k = 3$. The differential module accepts the selection under test when at least k reference executions reproduce it. All three thresholds use the same recorded full-pool, target-pool, and adaptive-pool executions from Section 4.2.

Table 13: Judgment reliability (%) when accepting the selection under test requires one, two, or three matching reference executions. A match means that a reference execution reproduces the selection under test.

Benchmark	Required matches	Accuracy	Precision	Recall	F1
SkillsBench	1 (default)	78.02	**100.00**	68.25	81.13
	2	86.81	**100.00**	80.95	89.47
	3	**93.41**	**100.00**	**90.48**	**95.00**
SkillRet	1 (default)	96.33	**100.00**	95.95	97.93
	2	98.02	**100.00**	97.82	98.90
	3	**98.59**	**100.00**	**98.44**	**99.22**
R3-Skill	1 (default)	80.00	**100.00**	61.50	76.16
	2	83.89	**100.00**	68.98	81.65
	3	**88.06**	98.65	**78.07**	**87.16**
SRA-Bench	1 (default)	86.07	**99.19**	71.51	83.11
	2	89.97	97.22	81.40	88.61
	3	**93.87**	96.88	**90.12**	**93.37**
Macro average	1 (default)	85.11	**99.80**	74.30	84.58
	2	89.67	99.31	82.29	89.66
	3	**93.48**	98.88	**89.28**	**93.69**

Increasing the matching threshold improves accuracy, recall, and F1 on every benchmark (Table 13). From $k = 1$ to $k = 3$, the largest recall gain occurs on SkillsBench, from 68.25% to 90.48%, while the smallest occurs on SkillRet, from 95.95% to 98.44%. Precision remains at 100% on SkillsBench and SkillRet, but decreases on R3-Skill and SRA-Bench. Among the 154 false negatives at $k = 1$, 45 have one matching reference execution and 40 have two. Raising the threshold to two or three therefore identifies 45 or 85 previously missed selections, respectively. The remaining 69 have three matching reference executions and therefore remain unreported even at $k = 3$. Higher thresholds also report deviations for some correct selections because too few reference executions reproduce them. Across the four benchmarks, false positives increase from one at $k = 1$ to four at $k = 2$ and seven at $k = 3$. We retain $k = 1$ in the discovery experiments to limit false-positive reports.

E Case Studies

Each execution case gives the original task, the selections from each pool, and the resulting judgment. In the tables, the Full row gives the selection under test; the Target and Adaptive rows give reference selections. $\varnothing$ denotes no selected skill. Tables group runs only when both their available and selected skills are identical. The task generation case compares generated requests and does not include agent executions.

E.1 Task Generation for Different Skill Uses

This case study compares the distribution of five tasks generated by each method from `sympy` in the SkillsBench task diversity experiment (Section 4.3). Both methods use `GPT-5.4-mini` at temperature 0.7. D1 through D5 represent direct generation, while F1 through F5 represent constraint-guided generation. Appendix F.3 provides excerpts from the relevant skill documentation and the full set of nine extracted constraint groups.

Table 14: Distribution of generated tasks across the extracted constraint groups for `sympy`. D1–D5 and F1–F5 denote tasks from direct and constraint-guided generation, respectively. Tasks are mapped by their requested core intents and may appear in multiple rows.

Group	Core intent: behavior / object / output	Direct	SKILLDIVER
g_1	solve / equations / exact symbolic solutions	D1, D2, D3, D5	—
g_2	perform / calculus operations / symbolic derivatives, integrals, limits, or series results	D2, D3, D4, D5	—
g_3	manipulate / algebraic expressions / simplified or transformed symbolic expressions	D1, D2, D3, D4	F2, F4
g_4	work with / matrices / symbolic matrix results such as inverses, determinants, eigenvalues, or solved linear systems	—	—
g_5	perform / physics calculations / symbolic physics results	—	F1
g_6	compute / number theory problems / exact symbolic number-theoretic results	—	—
g_7	compute / geometry problems / exact symbolic geometric results	—	F5
g_8	generate / executable code from mathematical expressions / code that can evaluate the expressions	—	F3
g_9	generate / formatted mathematical output / LaTeX or other formatted mathematical representation	—	F4

Across five tasks per method, direct generation covers three core intents and constraint-guided generation covers five (Table 14). The mean pairwise cosine distance is 0.3701 for direct generation and 0.5610 for constraint-guided generation.

E.2 AGREEMENT BETWEEN TARGET AND FULL POOLS

This case comes from the SkillsBench discovery experiment with Codex. The task is generated from `civ6lib`.

Task.

> In Civilization 6 with the Gathering Storm ruleset, can you check whether a district placement I'm considering is valid or invalid? I'm looking at a specific tile and want a clear placement validation.

Table 15: Selections for the Civ6 district-placement task. C denotes `civ6lib`.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{C\}$	$\{C\}$
Full	1	All 199 SkillsBench skills	$\{C\}$

Skill document (excerpt).

> Complete reference for Civilization 6 district placement and adjacency bonus systems (Gathering Storm).
>
> [...]
>
> **Usage**

```
# Validate a placement
rules = get_placement_rules(
    tiles, city_center=(21, 13), population=7)
result = rules.validate_placement(
    DistrictType.CAMPUS, 21, 14, {})
if result.valid:
    print("Valid placement!")
```

The request asks for placement validation under Gathering Storm, which the document explicitly supports.

E.3 Alternative Selection for Audio Conversion

This case expands the audio-conversion example in the Introduction. It comes from the SkillsBench discovery experiment with OpenHands and GLM-5.3-Flash. The task is generated from `FFmpeg Audio Processing`.

Task.

> Please re-encode this audio recording into a different format, like converting it from WAV to MP3, and give me the audio back in the new format.

Table 16: Selections for the audio-conversion task. A denotes `FFmpeg Audio Processing`, and C denotes `FFmpeg Format Conversion`.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{A\}$	$\{A\}$
Full	1	All 199 SkillsBench skills	$\{C\}$
Adaptive	1, 2, 3	$\{A, C\}$	$\{C\}$

The full-pool execution selects `FFmpeg Format Conversion`, which is unavailable in the target pool. All three adaptive-pool executions reproduce this selection, so the differential module reports no deviation. The execution traces record reads of the corresponding skill documents in all seven executions.

FFmpeg Audio Processing (excerpt).

> **When to Use**
>
> [...]
>
> - Convert audio formats
>
> [...]

FFmpeg Format Conversion (excerpt).

> **When to Use**
>
> [...]
>
> - Convert audio formats (MP3, AAC, WAV, etc.)
>
> [...]
>
> **Audio Format Conversion**
>
> ```
> # MP3
> ffmpeg -i input.wav -acodec libmp3lame -q:a 2 output.mp3
> ```

Both skill documents support the requested conversion.

The adaptive pool allows the agent to select either skill and reproduces an alternative supported by its documentation. Comparing against the target pool alone would report this selection as a deviation.

E.4 Reported Deviation for NPZ Validation

This SkillsBench case with Codex shows the NPZ-validation behavior illustrated in Figure 1. The task is generated from `output-validation`.

Task.

> Please check the NPZ mask file I generated from sampled video frames and verify that it is internally consistent with the frame coverage and array dimensions, so I can be sure the mask data is structurally consistent and complete.

Table 17: Selections for the NPZ-validation task. V denotes `output-validation`, and M denotes `dyn-object-masks`.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{V\}$	$\{V\}$
Full	1	All 199 SkillsBench skills	$\{M\}$
Adaptive	1, 2, 3	$\{V, M\}$	$\{V\}$

The full-pool execution selects `dyn-object-masks`, but all six reference executions select `output-validation`. Even when both skills are available in the adaptive pool, the agent selects only `output-validation`. None of the reference executions reproduces the selection under test, so the differential module reports a deviation.

output-validation (excerpt).

> **When to use**
>
> - After generating your outputs (interval instructions, masks, etc.), before submission/hand-off.
>
> **Checks**
>
> [...]
>
> - **Coverage**: max frame index $\leq$ video total-1; consistent with your sampling policy.
> - **Frame count**: NPZ `f_{i}_*` count equals sampled frame count; no gaps or missing components.
> - **CSR integrity**: each frame has `data/indices/indptr`; `len(indptr)==H+1`; `indptr[-1]==indices.size`; indices within `[0,W)`.
> - **Value validity**: JSON values are non-empty string lists; labels in the allowed set.

dyn-object-masks (excerpt).

> **When to use**
>
> - Detect moving objects in scenes with camera motion; produce sparse masks aligned to sampled frames.

As in Task A of Figure 1, the request asks to validate an existing NPZ file, matching the checks documented by `output-validation`. The `dyn-object-masks` skill is used to detect moving objects and produce sparse masks aligned to sampled frames. SKILLDIVER also generates a JSON-validation request from the `output-validation` document in the task diversity experiment:

> Please check the JSON interval instructions I'm using with sampled frame outputs in my video-processing pipeline and confirm that the instruction keys and values are all valid.

This request illustrates the second documented scenario in Figure 1.

E.5 Examples of Selection Deviation Types

These cases come from the SkillsBench discovery experiment with Codex and expand the analysis in Section 4.1. They illustrate mixed, omission, and addition deviations relative to the reference selections.

E.5.1 Mixed Deviation

Task.

> I'm working in the library codebase and need a clear intended-contracts view of how the trainer hierarchies fit together, including the shared utilities, configuration objects, model wrappers, and how data flows through them. Please trace those pieces through the existing code and explain the expected responsibilities and interfaces in a way I can use to understand the library's current structure.

Table 18: Selections for the trainer-hierarchy task. T denotes `trl`, and D denotes `discover-important-function`.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{T\}$	$\{T\}$
Full	1	All 199 SkillsBench skills	$\{D\}$
Adaptive	1, 2, 3	$\{T, D\}$	$\{T\}$

trl (excerpt).

> **Description**
> Reference for the TRL (Transformer Reinforcement Learning) library codebase. [...] Covers trainer hierarchy (SFT, DPO, GRPO, KTO), shared utility functions (`selective_log_softmax`, `decode_and_strip_padding`, padding helpers), configuration system, model wrappers, and how data flows through any TRL trainer.

discover-important-function (excerpt).

> **Description**
> When given a project codebase, this skill observes the important functions in the codebase for future action.
> **Purpose**
> This skill helps an agent quickly narrow a Python repository down to a small set of high-value fuzz targets.

The `trl` skill documents trainer hierarchies, shared utilities, and data flow, matching the requested codebase analysis. The `discover-important-function` skill focuses on identifying functions to fuzz. The full-pool execution reads `discover-important-function` and does not select `trl` after recognizing the fuzzing focus. All six reference executions select `trl`, so the module reports both an omission of `trl` and an addition of `discover-important-function`.

E.5.2 Omission Deviation

Task.

> Can you send a message from one process on my cluster to a process running on another node, and make sure the remote process receives it?

Table 19: Selections for the interprocess-message task. E denotes `erlang-distribution`; $\varnothing$ means that no skill is selected.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{E\}$	$\{E\}$
Full	1	All 199 SkillsBench skills	$\varnothing$

erlang-distribution (excerpt).

> **Distributed Message Passing**
> Send messages to processes on remote nodes using same syntax as local messaging.
>
> ```
> send_remote(Node, Name, Message) ->
>     {Name, Node} ! Message.
> ```

The `erlang-distribution` skill provides Erlang examples for sending messages to remote processes and waiting for replies, supporting the task's request for message delivery and receipt confirmation. All three target-pool executions select this skill, while the full-pool execution selects none, so the module reports an omission.

E.5.3 ADDITION DEVIATION

Task.

> I'm working on earthquake cataloging and want a clear comparison of seismic event detection and phase picking methods. Please summarize the main approaches, their tradeoffs, strengths, and limitations so I can decide which ones are most appropriate for building a reliable catalog.

Table 20: Selections for the seismic-method comparison. P denotes `seismic-picker-selection`, and S denotes `seisbench-model-api`.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{P\}$	$\{P\}$
Full	1	All 199 SkillsBench skills	$\{P, S\}$
Adaptive	1, 2, 3	$\{P, S\}$	$\{P\}$

seismic-picker-selection (excerpt).

> **Description**
> This is a summary the advantages and disadvantages of earthquake event detection and phase picking methods, shared by leading seismology researchers at the 2025 Earthquake Catalog Workshop. Use it when you have a seismic phase picking task at hand.

seisbench-model-api (excerpt).

> **Description**
> An overview of the core model API of SeisBench, a Python framework for training and applying machine learning algorithms to seismic data. It is useful for annotating waveforms using pretrained SOTA ML models, for tasks like phase picking, earthquake detection, waveform denoising and depth estimation. [. . .]

The `seismic-picker-selection` skill summarizes the strengths and limitations of earthquake detection and phase-picking methods, matching the requested comparison. The `seisbench-model-api` skill provides guidance on applying pretrained models to waveform data. All six reference executions select only `seismic-picker-selection`, while the full-pool execution selects both skills, so the module reports an addition.

E.6 A SELECTION ERROR MISSED BY THE LLM JUDGE

In this SRA-Bench case, the differential module identifies an incorrect selection that the LLM judge accepts (Section 4.2).

Task.

> Which postal code region has most bussinesses in Upper Gwynedd, PA?

Table 21: Codex selections for the postal-code query. Y denotes `Yelp aggregation`, and W denotes `Web Search`. The benchmark skill set is $\{Y\}$.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{Y\}$	$\{Y\}$
Full	1	200 candidate skills	$\{W\}$
Adaptive	1, 2, 3	$\{Y, W\}$	$\{Y\}$

The `Yelp aggregation` skill supports SQL aggregation over the local Yelp database, matching the task's request to compare business counts by postal code. All six reference executions select

this skill, while the full-pool execution selects `Web Search`. Therefore, the differential module reports the selection deviation. The LLM judge receives the request and the complete document for the selected `Web Search` skill (`web_11588`), whose description begins "DEFAULT for all research and web queries." It returns "yes", accepting that selection. Relative to the benchmark label, this is a false negative for the LLM judge and a true positive for the differential module.

E.7 Judgment Error Cases

The following Codex cases examine judgment errors against benchmark labels, as analyzed in Appendix D.5.

E.7.1 False Negative

In this SRA-Bench case, one reference execution repeats a selection that omits a benchmark-labeled skill.

Task.

> Perform DBSCAN clustering on the data by transforming it into a DataFrame and recording the clusters in a new column named 'Cluster'. Please choose the parameters `eps=3` and `min_samples=2`.
>
> The function should output with: `df` (DataFrame): The DataFrame with a new 'Cluster' column.
>
> You should write self-contained code starting with:
>
> ```
> import pandas as pd
> from sklearn.cluster import DBSCAN
> def task_func(data, cols):
> ```

Table 22: Selections for the DBSCAN clustering task. P denotes the `pandas` skill, and S denotes the `scikit-learn` skill. The benchmark skill set is $\{P, S\}$.

Pool	Run(s)	Available skills	Selected skills
Target	1	$\{P, S\}$	$\{S\}$
Target	2, 3	$\{P, S\}$	$\{P, S\}$
Full	1	200 candidate skills	$\{S\}$

The benchmark skill set includes `pandas` for DataFrame operations and `scikit-learn` for clustering. The full-pool execution selects only `scikit-learn`, as does the first target-pool execution. The single matching reference execution causes the default rule to accept the selection under test, missing the omission of `pandas` relative to the benchmark skill set.

E.7.2 False Positive

The only false positive is SRA-Bench task `champ_00020`. The benchmark skill set contains *Rule of Sum* (`champ_012`) and *Rule of Product* (`champ_013`).

Task.

> Find the number of strings consisting of 5 digits of 1, 2 and 3 where no two 1s are neighbors.

Table 23: Selections for the digit-string counting task. S denotes *Rule of Sum*, and P denotes *Rule of Product*. The benchmark skill set is $\{S, P\}$.

Pool	Run(s)	Available skills	Selected skills
Target	1, 2, 3	$\{S, P\}$	$\{S\}$
Full	1	200 candidate skills	$\{S, P\}$

The *Rule of Sum* skill supports adding the counts for different numbers of 1s, while *Rule of Product* supports counting the ways to fill the remaining positions with 2 or 3. The full-pool execution selects both skills and matches the benchmark skill set. However, all three target-pool executions omit *Rule of Product*, so none reproduces the selection under test and the differential module incorrectly reports a deviation.

F Method Details and Evaluation Prompts

The procedure, prompts, and examples below supplement the method in Section 3. This appendix also gives the baseline and evaluator prompts for Section 4.

F.1 Adaptive Differential Testing

Algorithm 1 shows the adaptive differential testing procedure for one task. The collection $\mathcal{Y}$ contains the reference selections, and Y_{full} is the selection under test.

Algorithm 1 Adaptive differential testing for one task

Require: Task x, Agent A, Skill Chain $\tau \subseteq P_{\text{full}}$, Reference Counts $R_{\text{tar}}, R_{\text{adp}}$

$P_{\text{tar}} \leftarrow \tau, \mathcal{Y} \leftarrow \emptyset$
for $r = 1, \ldots, R_{\text{tar}}$ **do**
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\text{Sel}(A, x, P_{\text{tar}})\}$
end for
$Y_{\text{full}} \leftarrow \text{Sel}(A, x, P_{\text{full}})$
if $Y_{\text{full}} \setminus P_{\text{tar}} \neq \emptyset$ **then**
  $P_{\text{adp}} \leftarrow P_{\text{tar}} \cup Y_{\text{full}}$
  for $r = 1, \ldots, R_{\text{adp}}$ **do**
    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\text{Sel}(A, x, P_{\text{adp}})\}$
  end for
end if
return $\mathbb{I}[Y_{\text{full}} \notin \mathcal{Y}]$

F.2 Prompts of SkillDiver

Figure 4 gives core intent extraction, Figure 5 gives context and boundary extraction for a fixed intent, and Figure 6 gives task generation from sampled constraints. Each prompt includes its input JSON, with angle brackets marking values supplied for the current skill or task. To account for potential sequential dependencies between skills, the input of task generation preserves the GoS order via step indices, thereby enhancing task coherence.

Core intent extraction

System message

You are a skill-selection core-intent extractor. Extract independently requestable behavior-object-output cores from one skill document. Each core must bind one action, its target object, and the direct user-visible result of that action. Use only the structured skill JSON as evidence. Return only the required JSON object; do not include markdown, commentary, rationale, source spans, or hidden reasoning.

User message

<task>
Extract independently requestable behavior-object-output cores for skill selection.
</task>

<input>
Structured skill JSON:
```
{
  "skill_name": "<skill name>",
  "description": "<skill description>",
  "source_text": "<complete skill document>"
}
```
</input>

<instructions>
- Treat description as the authoritative definition of the skill's selection scope.
- Derive candidate cores from capabilities explicitly named in the description.
- Use source_text only to clarify wording or distinguish outcomes already supported by the description; a broad description term does not license splitting every implementation action found in source_text.
- Each core contains one canonical behavior, its directly associated object, and one canonical direct output.
- The output must be the immediate user-visible result of performing the behavior on the object.
- If materially different outputs imply different requested operations or final states, split them into separate cores.
- Merge output paraphrases that describe the same final state.
- Keep cores separate when the description presents them as independently requestable capabilities with materially different requested outcomes.
- Preserve distinct description-level actions such as create, edit, delete, analyze, extract, comment, accept changes, or reject changes when each is supported.
- Merge true behavior paraphrases such as edit, modify, and change when they produce the same output on the same object state.
- Prefer the description-level intent over lower-level means used to implement it. For example, move, rename, archive, or delete are not separate cores when they only implement a broader cleanup or organization capability.
- Do not treat formatting, preservation, safety, approval, or implementation properties as independent outputs.
- Do not promote instructions, workflow stages, tool mechanics, examples, or intermediate actions from source_text into cores.
- Do not include context, boundaries, implementation steps, workflows, or examples.
- Do not omit a distinct description-level core merely to reduce the number of cores.
</instructions>

<response_format>
Return exactly one JSON object matching this schema:
```
{
  "core_intents": [
    {
      "behavior": "one canonical user-request action",
      "object": "the directly associated target object",
      "output": "the direct user-visible result of that action on that object"
    }
  ]
}
```
</response_format>

Figure 4: Complete core intent extraction prompt. Angle brackets mark values supplied for each skill.

Context and boundary extraction

System message

You are a skill-selection context extractor. Given one fixed behavior-object-output core, extract only task-specific context values and explicit routing boundaries compatible with that core. Use only the provided structured skill JSON as evidence. Return only the required JSON object; do not include markdown, commentary, rationale, source spans, or hidden reasoning.

User message

<task>
Extract task-specific contexts and boundaries for the provided core.
</task>

<input>
Core intent JSON:
```
{
  "behavior": "<fixed behavior>",
  "object": "<fixed object>",
  "output": "<fixed output>"
}
```
Structured skill JSON:
```
{
  "skill_name": "<skill name>",
  "description": "<skill description>",
  "source_text": "<complete skill document>"
}
```
</input>

<instructions>
- The behavior, object, and output are fixed. Do not introduce another action, target object, or final result.
- Treat description as the authoritative scope and use source_text only to add details compatible with that scope.
- Context values instantiate this core as concrete user tasks: a domain, artifact instance, environment, scope, precondition, or user situation.
- Every context must be naturally expressible by a user and must not change the fixed behavior-object-output semantics.
- Do not use internal routing criteria, expected tool-call counts, skill activation rules, agent workflow stages, storage mechanics, or implementation instructions as context.
- Boundary values describe an adjacent request scope that should not be treated as this core, such as create versus edit or analyze versus modify.
- Do not use safety, preservation, approval, formatting, logging, or quality instructions as boundaries.
- Do not copy workflow checklists or enumerate incidental examples, file extensions, locations, metadata fields, or subtypes that do not create meaningfully different task contexts.
- Return an empty list for an unsupported category. Do not guess from general knowledge.
- Merge true paraphrases, but do not omit distinct task contexts merely to reduce the list size.
</instructions>

<response_format>
Return exactly one JSON object matching this schema:
```
{
  "context": [
    "compatible task-specific setting, scope, or user scenario"
  ],
  "boundary": [
    "explicit grounded exclusion for this core intent"
  ]
}
```
</response_format>

Figure 5: Complete context and boundary extraction prompt. Each call supplies one fixed core intent and its skill document. Angle brackets mark the supplied values.

Task generation

System message

You are a user-input generator for skill-selection testing.
Your task is to write one realistic end-user request that concretizes the provided semantic requirements.
For multi-step requests, use the per-step trigger semantics and handoff relations in the <input> section.
Use only the information in the <input> section; do not add unrelated tasks, tools, frameworks, or goals.
Do not mention internal skill identifiers, chain names, oracle labels, routing hints, evaluator terms, dimension names, edge ids, relation types, constraint ids, or field names.
Normal domain terms are allowed when they are necessary to express the user's task.
Return only the JSON object required by <response_format>; do not include markdown or commentary.

User message

<task>
Generate one realistic end-user request from the provided step-level semantic requirements.
</task>

<input>
Generation JSON:

```
{
  "composition": {
    "steps": [
      {
        "step_index": 1,
        "trigger_semantics": {
          "behavior": ["<selected behavior>"],
          "object": ["<associated object>"],
          "context": ["<sampled context>"],
          "output": ["<associated output>"]
        },
        "must_avoid": {
          "boundary": ["<sampled boundary>"]
        }
      }
    ]
  },
  "generation_instructions": [
    "Write a concrete, realistic user request, not a summary of the constraints.",
    "Use composition.steps as the only source of required task semantics.",
    "Naturally express each step's behavior, object, context, and output.",
    "For multi-step tasks, express the steps as one connected user request.",
    "Do not ask for any step boundary or out-of-scope task.",
    "Do not mention skill names, routing labels, oracle metadata, graph edges, or internal identifiers.",
    "Normal domain terms are allowed when they are part of the object, context, or output."
  ]
}
```

</input>

Figure 6: Task generation prompt, continued on the next page. The input shows one step. For multiple skills, the steps array contains one entry per skill. Angle brackets mark sampled values, and constraint lists may be empty.

Task generation (continued)

User message (continued)

<instructions>
- Use composition.steps as the only source of required task semantics. A single-skill request has one step; a composition request has multiple steps.
- For each step, preserve the provided behavior-object-output relationship and express every non-empty context value through a concrete but plausible user scenario.
- Treat each step's must_avoid.boundary values as adjacent out-of-scope content that must not be requested.
- For multi-step requests, express the step order naturally with first/then/after language when useful.
- Do not copy the constraints mechanically. Turn them into a natural user request with concrete task-specific details.
- Concrete details must support the given core and context. Do not introduce unrelated tools, frameworks, products, domains, outputs, or extra tasks.
- Do not mention skill identifiers, chain ids, edge ids, relation types, oracle metadata, routing/evaluator terms, dimension names, constraint ids, or field names such as trigger_semantics, must_avoid, or composition.
</instructions>

<response_format>
Return exactly one JSON object with this shape:

```
{
  "prompt": "<user_request>"
}
```

</response_format>

Figure 6 (continued): Remaining instructions and response format in the same user message.

F.3 Constraint Extraction Example

We use `sympy` to illustrate constraint extraction from a skill document.

Skill document (excerpt). The excerpt retains the original wording and headings, with bracketed ellipses marking omissions. The use list corresponds to the nine intents below, and the code examples distinguish executable functions from LaTeX output.

When to Use This Skill

Use this skill when:

- Solving equations symbolically (algebraic, differential, systems of equations)
- Performing calculus operations (derivatives, integrals, limits, series)
- Manipulating and simplifying algebraic expressions
- Working with matrices and linear algebra symbolically
- Doing physics calculations (mechanics, quantum mechanics, vector analysis)
- Number theory computations (primes, factorization, modular arithmetic)
- Geometric calculations (2D/3D geometry, analytic geometry)
- Converting mathematical expressions to executable code (Python, C, Fortran)
- Generating LaTeX or other formatted mathematical output
- Needing exact mathematical results (e.g., `sqrt(2)` not `1.414...`)

[...]

7. Code Generation and Output

Convert to executable functions:

```
from sympy import lambdify
import numpy as np

expr = x**2 + 2*x + 1
f = lambdify(x, expr, 'numpy')  # Create NumPy function
x_vals = np.linspace(0, 10, 100)
y_vals = f(x_vals)  # Fast numerical evaluation
```

[...]

LaTeX output:

```
from sympy import latex
latex_str = latex(expr)  # Convert to LaTeX for documents
```

Complete extracted constraints. The nine constraint groups extracted by SKILLDIVER from this document are shown below.

Group g_1: Equation solving.

Behavior b	solve
Object o	equations
Output r	exact symbolic solutions
Context candidates $\mathcal{C}$	(1) solving algebraic equations symbolically (2) working with equations that have variables and parameters (3) needing exact symbolic solutions rather than numerical approximations (4) solving systems of equations symbolically (5) solving differential equations symbolically
Boundary candidates $\mathcal{B}$	(1) tasks that are only numerical approximation of equation roots (2) non-mathematical problem solving (3) purely numeric computation without symbolic variables (4) tasks centered on calculus operations other than equation solving (5) tasks centered on matrices, physics, geometry, number theory, or code generation instead of equations

Group g_2: Calculus.

Behavior b	perform
Object o	calculus operations
Output r	symbolic derivatives, integrals, limits, or series results
Context candidates $\mathcal{C}$	(1) symbolic mathematics in Python (2) exact calculus computation with variables or parameters (3) derivatives, integrals, limits, or series expansions needing symbolic results (4) mathematical formulas expressed algebraically rather than numerically
Boundary candidates $\mathcal{B}$	(1) numerical approximation or floating-point-only computation (2) non-calculus symbolic tasks outside algebraic manipulation or exact symbolic math (3) tasks centered on matrices, physics, number theory, geometry, or code generation instead of calculus operations

Group g_3: Algebraic manipulation.

Behavior b	manipulate
Object o	algebraic expressions
Output r	simplified or transformed symbolic expressions
Context candidates $\mathcal{C}$	(1) symbolic mathematics in Python (2) exact algebraic manipulation of expressions with variables or parameters (3) simplifying or transforming symbolic formulas for a user
Boundary candidates $\mathcal{B}$	(1) numeric approximation tasks instead of exact symbolic results (2) non-symbolic programming tasks that do not involve mathematical expressions (3) creating or formatting output without manipulating algebraic expressions

Group g_4: Matrix operations.

Behavior b	work with
Object o	matrices
Output r	symbolic matrix results such as inverses, determinants, eigenvalues, or solved linear systems
Context candidates $\mathcal{C}$	(1) symbolic matrix computation in Python (2) exact linear algebra tasks involving matrices (3) user needs symbolic results for matrix problems with variables or parameters (4) matrix operations such as inverses, determinants, eigenvalues, eigenvectors, diagonalization, transposes, and solving linear systems (5) symbolic manipulation of matrices rather than numerical approximation
Boundary candidates $\mathcal{B}$	(1) non-matrix symbolic mathematics such as calculus or algebraic equation solving (2) purely numerical matrix computation or approximate linear algebra results (3) physics, geometry, number theory, or other advanced math tasks not centered on matrices (4) code generation, LaTeX output, or formatting tasks when not primarily about matrix results (5) general symbolic expression simplification that does not involve matrices

Group g_5: Physics calculations.

Behavior b	perform
Object o	physics calculations
Output r	symbolic physics results
Context candidates $\mathcal{C}$	(1) symbolic physics calculations in Python (2) exact symbolic results for physics formulas with variables and parameters (3) classical mechanics, quantum mechanics, or vector-analysis physics problems requiring symbolic computation (4) physics calculations that benefit from algebraic manipulation, solving equations, calculus, or matrix methods
Boundary candidates $\mathcal{B}$	(1) purely numerical physics approximation tasks (2) non-physics symbolic math tasks that are not about physics calculations (3) requests focused only on generating code or formatted output without performing physics calculations (4) physics tasks that require only measurement handling or data analysis rather than symbolic results

Group g_6: Number theory.

Behavior b	compute
Object o	number theory problems
Output r	exact symbolic number-theoretic results
Context candidates $\mathcal{C}$	(1) symbolic number theory computation in Python (2) exact arithmetic tasks involving primes, factorization, gcd/lcm, modular arithmetic, or Diophantine equations (3) math problems where the user wants exact symbolic results rather than numeric approximations (4) advanced mathematical problem solving within a symbolic computation environment
Boundary candidates $\mathcal{B}$	(1) non-symbolic or purely numerical approximation tasks (2) general mathematics outside number theory when no symbolic computation is needed (3) creating or converting mathematical expressions to executable code as the primary goal (4) physics, calculus, matrix, or geometry tasks not framed as number theory problems

Group g_7: Geometry.

Behavior b	compute
Object o	geometry problems
Output r	exact symbolic geometric results
Context candidates $\mathcal{C}$	(1) symbolic geometry computation in Python (2) analytic geometry problems requiring exact symbolic results (3) geometry tasks involving variables or parameters where closed-form results are needed (4) 2D or 3D geometry calculations using symbolic mathematics
Boundary candidates $\mathcal{B}$	(1) purely numerical geometry approximation tasks (2) non-geometry symbolic mathematics tasks (3) geometry tasks that require only code generation or formatted output rather than computation (4) physics, algebra, calculus, matrix, number theory, or other non-geometry problem domains

Group g_8: Code generation.

Behavior b	generate
Object o	executable code from mathematical expressions
Output r	code that can evaluate the expressions
Context candidates $\mathcal{C}$	(1) generating executable Python code from mathematical formulas or expressions (2) converting symbolic expressions with variables or parameters into a runnable function for evaluation (3) creating code for exact symbolic mathematics in Python using SymPy (4) working with mathematical expressions that need executable evaluation rather than plain algebraic output
Boundary candidates $\mathcal{B}$	(1) requests that only ask to simplify, solve, differentiate, integrate, or otherwise manipulate expressions without generating code (2) tasks focused on numerical approximation or direct numeric answers instead of executable code (3) requests to generate non-code mathematical output such as LaTeX or pretty-printed formulas (4) modifying or debugging existing code that is not primarily about converting mathematical expressions into executable evaluation code

Group g_9: Mathematical formatting.

Behavior b	generate
Object o	formatted mathematical output
Output r	LaTeX or other formatted mathematical representation
Context candidates $\mathcal{C}$	(1) symbolic mathematics tasks in Python (2) exact algebraic computation with variables or parameters (3) solving equations algebraically or as systems (4) calculus operations such as derivatives, integrals, limits, or series (5) manipulating, simplifying, expanding, factoring, or collecting expressions (6) symbolic matrix and linear algebra computations (7) geometry, number theory, combinatorics, logic, set theory, statistics, or physics calculations expressed symbolically (8) generating LaTeX or other formatted mathematical representations from symbolic expressions
Boundary candidates $\mathcal{B}$	(1) purely numerical approximation without symbolic exactness (2) non-mathematical content generation (3) tasks that do not involve symbolic mathematics (4) general document formatting unrelated to mathematical expressions

F.4 Task Generation Example

In this example, SkillDiver samples group g_3 with the context and boundary shown in Table 24.

Table 24: Constraint inputs for task generation from `sympy`. The core intent comes from g_3, with one context and one boundary sampled from that group.

Field	Value
Behavior b	manipulate
Object o	algebraic expressions
Output r	simplified or transformed symbolic expressions
Sampled context	simplifying or transforming symbolic formulas for a user
Sampled boundary	creating or formatting output without manipulating algebraic expressions

Based on these constraints, SKILLDIVER generates the task:

> Can you simplify this algebraic expression for me and show the transformed result: $(3x + 2) - 4(x - 5) + 2x$?

This is task F2 in Table 14.

F.5 BASELINE AND EVALUATOR PROMPTS

The following figures reproduce the prompts for GoS-Random-Direct baseline (Section 4.1), LLM judge baseline (Section 4.2), and naturalness evaluation (Section 4.3).

GoS-Random-Direct baseline. Figure 7 gives the direct generation prompt used by GoS-Random-Direct. The task diversity baseline and the w/o Constraint Guidance ablation also use it. The prompt uses complete skill documents to generate one task.

Direct task generation

System message

You generate benchmark user tasks for skill-selection testing. Use the provided SKILL.md documents as the input evidence. Return one strict JSON object and no other text.

User message

Generate exactly one natural user request that requires all target skills.
Do not mention skill names, SKILL.md, benchmark, baseline, or evaluation.
Do not include solution steps or expected skill labels in the task text.
Return exactly this JSON schema:
{"task":"<one user request>"}

Target skills: {target_skills_json}

Target skill documents:
{skill_documents}

Each block in `{skill_documents}`

<skill name="{skill_name}">
{complete_skill_document}
</skill>

Figure 7: Direct task generation prompt.

LLM judge baseline. Figure 8 shows the prompt used by the LLM judge. The `selected_skills` input concatenates one block per selected skill, separated by newlines. Each block contains the skill identifier, description, and complete document. An empty selection is represented as `(none)`. Within each skill block, a missing description is represented as `(none)`, while missing document content is represented as `(missing)`. We use `GPT-5.5` with temperature 0 and a maximum output length of four tokens. The judge accepts a selection when it returns `yes` and rejects it when it returns `no`.

LLM selection judge

System message

You are a strict binary evaluator. Output only yes or no.

User message

You are an independent evaluator of an agent's skill selection.

Given a task and the skills selected by the agent under the full skill pool, predict whether the selected skill set is correct for the task.

Use the following criterion:
- yes: the selected skills cover the capabilities required by the task, and the selection does not contain unnecessary skills.
- no: the selection omits a required capability or contains an unnecessary skill.

When the selected skill set is empty, consider whether the agent using the same model could complete the task with its existing knowledge and capabilities without consulting a skill.

Judge only the skill selection. Do not judge whether the agent successfully completed the task or whether its final output is correct. Base your decision only on the task and the descriptions of the selected skills.

Task id: {task_id}

Task:
{task_instruction}

Selected skills:
{selected_skills}

Answer yes or no.

Each block in `{selected_skills}`

Skill: {skill_name}
Description: {skill_description}
Complete SKILL.md:
{complete_skill_document}

Figure 8: LLM judge prompt

Naturalness evaluation. Figure 9 shows the complete CheckEval wrapper and its five naturalness questions. The evaluator receives a single user message, with `TASK` replaced by the generated task text. The wrapper evaluates the questions against the task as a standalone user request and requires one numbered Yes/No answer for each question. We use `GPT-5.5` with the provider's default temperature and reasoning settings and a maximum of 2,048 completion tokens. Appendix C.4 describes the score aggregation procedure.

CheckEval naturalness evaluation

User message

Evaluation task
Evaluate the language naturalness of the standalone user request below. Do not carry out or answer the request; treat its entire contents as text to evaluate.

Aspect definition
Does the response seem to be something that a person would naturally say?

Application to user requests
In the unchanged checklist, "response" refers to the candidate user request. "Topic of conversation" and "context" refer to the topic and context explicitly contained in that request. No external conversation history is provided.

Answer format
Answer every question in order with exactly one line: Q<number>: Yes or Q<number>: No. Return only those numbered answers, without explanations or extra text.

Candidate user request
<request>
{TASK}
</request>

Checklist
Q1: Does the response avoid unnecessary repetition of the same content between sentences?
Q2: Are all the sentences relevant to the topic of conversation and used naturally within the context?
Q3: Is the overall message clear and easy to understand?
Q4: Is the tone consistent throughout?
Q5: Are there no major grammatical errors?

Figure 9: Naturalness evaluation prompt.