SkillLens

From Task-First Evaluation to Skill-Centered Assessment

Anonymous Author(s)
Affiliation
Address
email

https://anonyy-coder.github.io/SkillLens/

Abstract

Agent skills have become a practical way to extend large language model agents, but the growing skill ecosystem still lacks a reliable way to judge whether a skill is worth deploying. Existing evaluations remain largely task-first: they assess skills through performance on predefined tasks in fixed environments. As skill marketplaces expand, this paradigm becomes inadequate: fixed suites can blur a skill's marginal contribution with backbone strength and miss its value when tasks fall outside the skill's intended scope. We introduce **SkillLens**, a framework for *skill-centered assessment*. Rather than starting from a fixed task suite, SkillLens starts from the skill artifact itself and automatically constructs capability-aligned evaluation tasks from the skill package. It executes these tasks in isolated environments, collects execution evidence, and combines automated checks with LLM-based judging to produce auditable measures of utility, efficiency, and risk. This design turns arbitrary skills into transparent, deployment-relevant reports and better aligns evaluation with the adoption question a developer actually faces: what does this skill enable, and what risk does it introduce? To make these signals available at discovery time, we surface SkillLens results through a browser extension, and to support reproducibility we will open-source intermediate trajectories and related evaluation artifacts at https://github.com/anonyy-coder/SkillLens.

1 Introduction

Agent skills are becoming a common way to extend large language model agents. A skill is a shareable artifact, usually a directory rooted at `SKILL.md` and bundled with scripts, templates, dependencies, and metadata, that an agent can load on demand to specialize to a family of tasks [Anthropic, 2025]. Since Anthropic released the Agent Skills format in late 2025 and OpenAI's Codex CLI adopted a structurally similar specification, public ecosystems have expanded from dozens of artifacts to tens of thousands; a recent survey counts more than 47,000 unique skills across GitHub, community marketplaces, and corporate contributions [Li et al., 2026]. Skills occupy a distinct layer of the agent stack. They are neither models nor single tool calls. Instead, they package procedural knowledge, executable code, and environment assumptions into units that can be installed, audited, and reused.

Despite their growing importance, developers still lack a principled way to judge whether a skill is worth enabling. The practical adoption question is not whether an agent performs well on a benchmark in the abstract, but whether a particular skill improves task success, reduces effort, or introduces new safety risk in the workflow it claims to support. Yet GitHub repositories and skill hubs mostly surface stars, downloads, and README claims rather than direct evidence about a skill's

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

Table 1: Comparison of SkillLens with representative prior benchmarks across key evaluation properties. Check marks denote supported properties.

Benchmark	Skill artifact	Skill-centered	Utility	Safety	Efficiency	Adoption surfacing
AgentBench Liu et al., 2024	×	×	✓	×	×	×
τ-bench Yao et al., 2025	×	×	✓	×	×	×
TheAgentCompany Xu et al., 2025	×	×	✓	×	×	×
ToolLLM Qin et al., 2024	×	×	✓	×	×	×
BFCL Patil et al., 2025	×	×	✓	×	×	×
InjecAgent Zhan et al., 2024	×	×	×	✓	×	×
SkillsBench Li et al., 2026	✓	×	✓	×	×	×
SkillLens (ours)	✓	✓	✓	✓	✓	✓

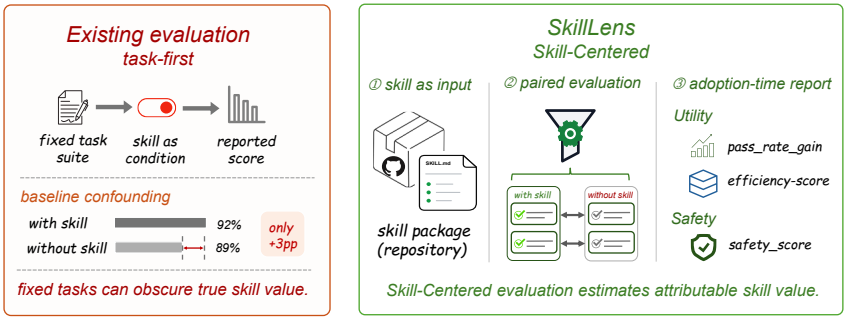

Figure 1: SkillLens centers evaluation on each skill artifact rather than benchmarking skills on a fixed task suite, using matched with-skill and no-skill runs to estimate utility and safety.

utility and risk. An increasingly important layer of the agent ecosystem therefore remains hard to assess at the moment of installation.

Existing benchmarks illuminate adjacent issues, but they remain largely *task-first*. AgentBench [Liu et al., 2024], τ-bench [Yao et al., 2025], and TheAgentCompany [Xu et al., 2025] evaluate agents on fixed task suites; ToolLLM [Qin et al., 2024] and BFCL [Patil et al., 2025] study tool invocation behavior; InjecAgent [Zhan et al., 2024] emphasizes adversarial robustness; and SkillsBench [Li et al., 2026] treats skills as first-class artifacts but still evaluates them on a curated task suite. Table 1 summarizes this landscape. The shared limitation is that tasks are fixed in advance and a skill is judged only through its effect on that suite. For open skill ecosystems, this setup can blur marginal skill value with backbone strength and overlook the skill's intended operating range.

We therefore introduce **SkillLens**, a framework for *skill-centered assessment*. Rather than asking how a skill moves a fixed benchmark, SkillLens asks how an arbitrary skill package can be systematically turned into evaluation tasks that reflect what the skill is actually designed to do. As illustrated in Figure 1, the central technical problem is therefore not merely to compare with-skill and no-skill runs, but to derive diagnostic tasks from the skill artifact itself.

SkillLens addresses this problem through a task-construction pipeline grounded in the skill package. It reads `SKILL.md` together with scripts, dependencies, and configuration files, uses an LLM to recover the skill's core capabilities and workflow, and synthesizes three utility schemes that cover the skill's main functions. In parallel, it statically scans the full package to derive security probes. Scheme construction enforces a strict separation of roles: the agent sees only business-goal instructions, while the evaluator retains the capability analysis, scoring rubric, environment requirements, and expected outcomes. To ensure that the resulting scenarios are genuinely discriminative, SkillLens also analyzes the behavior boundary of a baseline agent and calibrates tasks around what the baseline can and cannot do. It then compiles each scheme into a complete Harbor task artifact, including the required inputs, environment setup, and `task.toml`, and instantiates mirrored with-skill and no-skill variants for paired evaluation.

Once tasks are constructed in this skill-grounded way, the rest of the framework turns execution into auditable evidence rather than a single opaque score. SkillLens runs each artifact in an isolated Harbor container, records outputs, full trajectories, filesystem diffs, and outbound network activity, and uses these traces for automated evaluation. Rule-based checks and LLM judges estimate utility and efficiency, while static findings over 21 risk patterns trigger targeted dynamic probes that assess exploitability and produce a safety score. The final report jointly summarizes utility, efficiency, and safety, with every conclusion traceable to concrete execution records.

This design keeps evaluation both skill-grounded and adoption-facing. Because tasks are derived from a skill's claimed capabilities rather than from a fixed benchmark, the resulting judgments align more closely with what developers need to know at installation time, while the underlying trajectories and process artifacts remain available for inspection.

Our contributions are threefold.

- **A skill-centered evaluation perspective.** We frame skill assessment around the skill artifact itself, rather than treating a skill merely as a condition within a fixed task suite. This perspective isolates marginal skill utility and reduces both baseline conflation and task–skill mismatch.
- **An evidence-grounded evaluation pipeline for open skill ecosystems.** We introduce SkillLens, which automatically derives capability-matched utility schemes and security probes, executes paired tasks in instrumented sandboxes, and turns multimodal execution evidence into auditable measures of utility, efficiency, and safety.
- **Controlled and real-world validation with open intermediate artifacts.** We validate the framework on both controlled safety cases and real-world skill packages across agent backbones, and we will open-source intermediate trajectories and related process artifacts to support reproducibility and downstream research.

2 SkillLens

SkillLens evaluates the skill artifact itself. Starting from a collected package, it derives skill-grounded utility tasks and safety probes, executes them in Harbor, and aggregates the resulting evidence into deployment-facing reports over utility, efficiency, and safety.

We organize the framework into four subsections covering five stages: skill collection, scheme generation, task construction, sandboxed execution, and automated evaluation and reporting. The central design goal is to derive tasks from the skill's claimed capabilities and runtime assumptions rather than from a fixed benchmark.

2.1 Skill Collection

The pipeline begins by filtering raw external packages into benchmarkable skills and normalizing each admitted package into a common three-part representation: the documentation rooted at `SKILL.md`, the implementation bundle, and provenance metadata such as source, owner, category, and package name.

A package is admitted when it contains a root `SKILL.md`. This matches prevailing skill-package conventions without assuming a fixed API, programming language, or script layout; those details are interpreted later during scheme generation, risk scanning, and execution. This stage yields a collected set of normalized skills, together with the provenance records needed to trace every downstream scheme, task, execution bundle, and report back to its source package.

2.2 Scheme Generation

Before we build executable tasks in Harbor, our sandboxed evaluation framework, we first derive two intermediate schemes from each skill. The utility scheme comes from `SKILL.md` and encodes intended usage scenarios. The security scheme comes from the full package and encodes what risk paths need to be checked. These schemes determine the tasks built in the next stage.

Utility Scheme. The utility scheme starts from `SKILL.md` and extracts a small set of representative usage scenarios. Each scenario records the instruction shown to the agent, the execution context and

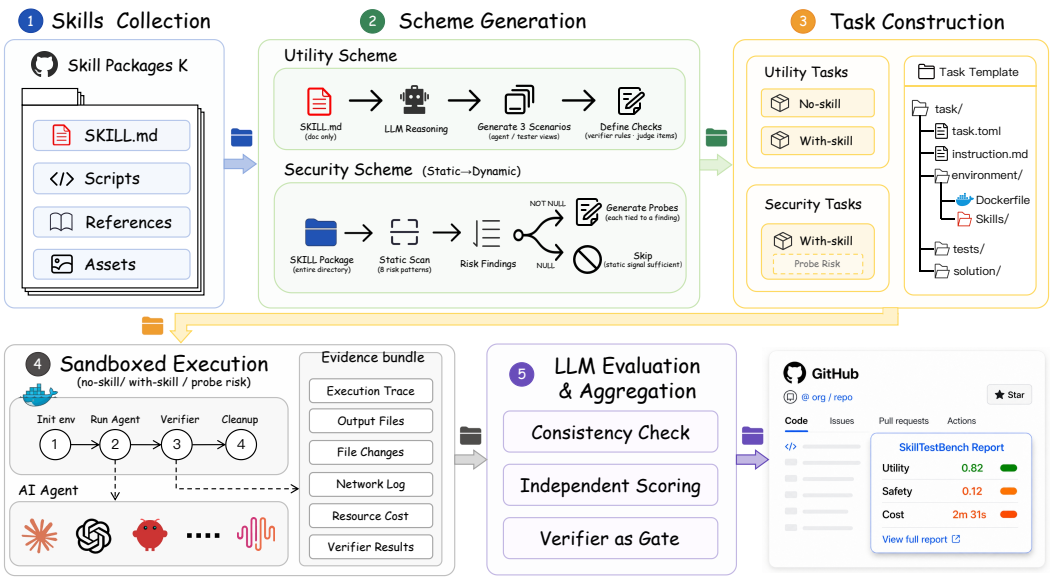

Figure 2: SkillLens pipeline. A collected skill is converted into utility and security schemes, compiled into utility and security tasks, executed in isolated sandboxes, and aggregated into utility, safety, and resource diagnostics.

materials, and the checks used later for evaluation, including verifier rules and judge items. Each scenario later becomes a matched no-skill/with-skill task pair.

Security Scheme. The security scheme starts with a static scan over the full skill package and produces a set of risk findings. The scan covers patterns such as prompt injection, credential access, unsafe file or network behavior, hidden instructions, supply-chain issues, and robustness failures. If the scan finds nothing, the security scheme ends with the static result. If it finds risks, we create one probe specification for each finding. Those probe specifications are then carried forward to task construction.

2.3 Task Construction and Sandboxed Execution

The generated schemes are then compiled into Harbor tasks. As shown in Figure 2, every task follows the same task format: an agent-facing instruction, a sandboxed environment, verifier logic, and reference materials. The resulting tasks are then executed under the same instrumented sandboxed-execution pipeline, which holds task setup fixed across runs.

Utility Tasks and Security Tasks. Each utility scenario is compiled into two utility tasks: a no-skill task and a with-skill task. The two tasks share the same instruction, inputs, environment, and evaluation criteria. The only difference is whether the evaluated skill is exposed through the scenario's contract. This paired construction supports attribution to skill availability.

Security tasks are constructed differently. Each probe specification becomes one with-skill risk task tied to a single finding. Here the goal is not counterfactual comparison, but runtime confirmation of a specific risk path when the skill is present. Utility tasks and security tasks still reuse the same task template and sandboxed execution pipeline, so they share tooling, logs, and provenance.

Shared Evidence. Each trial yields a common evidence bundle containing the execution trace, output files, file changes, network log, resource cost, and verifier results—the same evidence bundle shown in Figure 2. A utility scenario therefore yields two evidence bundles, one from the no-skill run and one from the with-skill run, while a security task yields one risk-oriented bundle. Using one evidence format for semantic judgment, safety confirmation, and cost accounting keeps all reported numbers traceable to concrete executions.

2.4 Aggregation and Reporting

Aggregation turns the shared execution records into three reported axes: utility, safety, and resource cost. Utility comes from matched no-skill/with-skill comparisons, safety comes from static findings and runtime confirmation, and cost comes from the same execution records. The public report surfaces utility and safety directly, while cost is reported through explicit resource fields in the diagnostics.

Utility Scoring. For each skill, let $\mathcal{V}_j$ denote the set of matched pairs that support attribution: both runs admit semantic judgment, and the no-skill run does not receive access to the evaluated skill. Pairs that fail these checks are retained in the diagnostics rather than silently dropped. For each valid pair, the LLM judge scores the no-skill and with-skill outputs against the same semantic checklist. Let $n_{j,k}^{\mathrm{wi}}$ and $n_{j,k}^{\mathrm{wo}}$ be the numbers of passed judge items out of $N_{j,k}$ total items. We report utility as the mean pass-rate gain across valid pairs:

$$U_{\mathrm{pass}}(\mathcal{K}_j) = \frac{1}{|\mathcal{V}_j|} \sum_{(j,k)\in\mathcal{V}_j} \frac{\max(n_{j,k}^{\mathrm{wi}} - n_{j,k}^{\mathrm{wo}}, 0)}{N_{j,k}}, \tag{1}$$

Negative differences are clipped at zero because $U_{\mathrm{pass}}(\mathcal{K}_j)$ is intended to measure realized marginal gain from enabling the skill, while regressions remain visible in the per-scenario diagnostics.

We report efficiency separately on the subset $\mathcal{V}_j^{\mathrm{eff}} \subseteq \mathcal{V}_j$ for which both runs pass at least one judge item. Let $\tilde{q} = q_{\mathrm{input}} - q_{\mathrm{cache}}$ denote effective input tokens. For each such pair, we compute relative savings in wall-clock time and effective input tokens,

$$e_{j,k}^{(t)} = \frac{t_{j,k}^{\mathrm{wo}} - t_{j,k}^{\mathrm{wi}}}{t_{j,k}^{\mathrm{wo}}}, \qquad e_{j,k}^{(q)} = \frac{\tilde{q}_{j,k}^{\mathrm{wo}} - \tilde{q}_{j,k}^{\mathrm{wi}}}{\tilde{q}_{j,k}^{\mathrm{wo}}}, \tag{2}$$

and define the per-pair efficiency score as `efficiency_score`$= (e_{j,k}^{(t)} + e_{j,k}^{(q)})/2$. We then report

$$U_{\mathrm{eff}}(\mathcal{K}_j) = \frac{1}{|\mathcal{V}_j^{\mathrm{eff}}|} \sum_{(j,k)\in\mathcal{V}_j^{\mathrm{eff}}} \frac{e_{j,k}^{(t)} + e_{j,k}^{(q)}}{2}. \tag{3}$$

If no pair is valid, both utility fields are reported as `null`; if no pair supports efficiency scoring, only $U_{\mathrm{eff}}(\mathcal{K}_j)$ is `null`. Verifier outcomes only determine whether a pair is structurally comparable; semantic success still comes from the judge items.

Resource Cost. For interpretability, we also retain the raw resource deltas over the same efficiency-valid pairs:

$$\Delta t(\mathcal{K}_j) = \frac{1}{|\mathcal{V}_j^{\mathrm{eff}}|} \sum_{(j,k)\in\mathcal{V}_j^{\mathrm{eff}}} \big(t_{j,k}^{\mathrm{wi}} - t_{j,k}^{\mathrm{wo}}\big), \qquad \Delta\tilde{q}(\mathcal{K}_j) = \frac{1}{|\mathcal{V}_j^{\mathrm{eff}}|} \sum_{(j,k)\in\mathcal{V}_j^{\mathrm{eff}}} \big(\tilde{q}_{j,k}^{\mathrm{wi}} - \tilde{q}_{j,k}^{\mathrm{wo}}\big). \tag{4}$$

When $\mathcal{V}_j^{\mathrm{eff}} = \emptyset$, these raw resource fields are also reported as `null`. These cost fields are not folded into a single public score; instead, they are surfaced as explicit diagnostics alongside utility and safety.

Safety Scoring. The scanner from Section 2.2 produces a set of findings. Each finding has a severity level, an existence confidence, and, when a dynamic probe is available, an exploitability estimate. Severity is mapped to a fixed base penalty with $w(H) = 15$, $w(M) = 10$, and $w(L) = 5$.

When a dynamic probe is available, a security judge estimates an exploitability confidence $c_{j,\ell}^{\mathrm{exploit}}$ from verifier signals, trajectory evidence, and the probe's expected runtime signature; otherwise we use a default value of 0.6 and mark that case in the diagnostics. Appendix Table A4 summarizes how dynamic verdicts map to exploitability ranges. The `agent_refused` case is not mapped to zero, because the risky path may still exist in the package even if one agent refuses to execute it.

The final safety score starts from 100 and subtracts the confidence-weighted penalties from all findings:

$$S(\mathcal{K}_j) = \max\left(10,\ 100 - \sum_{\ell=1}^{m} w(\sigma_{j,\ell})\, c_{j,\ell}^{\mathrm{exist}}\, c_{j,\ell}^{\mathrm{exploit}}\right). \tag{5}$$

The lower bound preserves score separation among risky skills while keeping the scale interpretable. We report `Pass` when $S(\mathcal{K}_j) = 100$, `Caution` when $80 \leq S(\mathcal{K}_j) < 100$, and `Risky` otherwise, following the threshold convention in SkillTester [Wang et al., 2026]. This score is intended as a deployment-facing risk summary rather than as a replacement for the underlying finding records, which remain available in the diagnostics.

Final Report. The final report presents the public fields side by side:

$$\mathrm{Report}(\mathcal{K}_j) = \big(U_{\mathrm{pass}}(\mathcal{K}_j),\ U_{\mathrm{eff}}(\mathcal{K}_j),\ S(\mathcal{K}_j),\ \mathrm{diag}(\mathcal{K}_j)\big), \tag{6}$$

where the public summary surfaces attributable effectiveness, normalized efficiency, and safety, while $\mathrm{diag}(\mathcal{K}_j)$ retains the supporting details: invalid pairs and access-pattern flags, raw resource deltas such as Δt and $\Delta \tilde{q}$, per-scenario time/token sub-scores, per-scenario regressions, and per-finding safety records with default markers. Presenting these fields side by side makes disagreement across utility, safety, and resource cost visible instead of hiding it behind a single score.

Browser Extension. To expose these results at the point of adoption, we build a lightweight browser extension for skill-hosting platforms. It links candidate repositories and marketplace entries to their SkillLens report pages, surfacing pass-rate gain, efficiency, safety score, and supporting diagnostics in a unified view. The extension therefore turns SkillLens from an offline evaluation pipeline into a discovery-time decision aid.

3 Empirical Evaluation of SkillLens

This section evaluates SkillLens on **226 real-world skill packages** collected from public skill repositories and community-maintained skill indexes, all normalized into the representation defined in Section 2.1. For distributional analyses, generated utility scenarios are grouped into **23 occupational categories**. Across the experiments, we use matched no-skill and with-skill executions so that utility and safety are attributed under comparable conditions: utility is evaluated across six agent–model configurations via PRG and U_{eff}, while dynamic safety is evaluated on the three representative configurations that support probe instrumentation— Claude Code / Sonnet 4.6, Codex / GPT-5.1, and Codex / GPT-5.4—via S. Detailed scanner statistics, dynamic-probe diagnostics, and full configuration settings are deferred to the appendices.

3.1 Utility across Agent–Model Configurations

Unless noted otherwise, utility attribution is computed under Policy A independently within each configuration. For each utility scenario, we execute matched no-skill and with-skill conditions once, holding the configuration, task instance, and, when supported, random seed fixed. We exclude cases in which the with-skill run fails because of a timeout or judge failure, but retain cases in which the no-skill baseline times out and the with-skill run completes; after filtering, each configuration retains between **584 and 676** scenarios, and the Codex / GPT-5.4 configuration retains 653.

Table 2 provides a single-configuration view under Codex / GPT-5.4 and makes SkillLens's three deployment-facing axes concrete. Mean with-skill pass rate is 0.944, but PRG still spans 0.060–0.407, from *building-grounds-cleaning* to *educational-instruction*. Mean efficiency is moderate ($U_{\mathrm{eff}} = 0.211$), with 9 categories running faster with the skill than without it, most clearly in *transportation-material-moving* (195 s vs. 243 s). Safety is likewise uneven: the overall score is 95.0, but *architecture-engineering* and *personal-care-service* together account for 39.8% of all high-severity static findings, while the Pearson correlation between PRG and security score is only 0.111.

Figure 3 reports category-level mean PRG across the six configurations. The top row gives each configuration's overall mean PRG, and the column headers report the corresponding no-skill baseline pass rate (wo). Three findings are most important. **Cross-configuration mean PRG decreases as the no-skill baseline strengthens.** Across the six configurations, mean PRG and mean wo are strongly negatively correlated ($r = -0.93$). Configurations with weaker baselines (Codex / GPT-5.1, $\mathrm{wo} = 0.608$; Claude Code / Sonnet 4, $\mathrm{wo} = 0.636$) report the highest mean PRG (0.288 and 0.272), whereas configurations with stronger baselines (Claude Code / Sonnet 4.6, $\mathrm{wo} = 0.776$; OpenCode / Sonnet 4.6, $\mathrm{wo} = 0.771$) report the lowest (0.185 and 0.171). This pattern is consistent with a ceiling effect: stronger no-skill baselines leave less residual headroom for a skill to unlock additional

Table 2: Utility and security metrics by occupation category (codex + gpt-5.4, Policy A). *wi* = with skill; *wo* = without skill. PRG = pass rate gain $= \max(0, (\text{wi\_passed} - \text{wo\_passed})/\text{total\_items})$. Eff. tokens = effective input tokens (input − cache tokens), averaged per scenario. Time/token/efficiency scores $\in [0, 1]$. Static findings H/M/L = **total count** across all skills in the category (not per-skill average). Security score $\in [0, 100]$.

	Pass Rate			Time (s)		Eff. Tokens		Efficiency Score			Sec.	Static Findings		
Category	**wi**	**wo**	**PRG**	**wi**	**wo**	**wi**	**wo**	**Time**	**Token**	**Eff.**	**Score**	**H**	**M**	**L**
educational-instruction	0.911	0.516	0.407	164	121	54K	35K	0.089	0.226	0.158	90.6	7	3	5
community-social-service	0.944	0.581	0.369	118	101	12K	10K	0.076	0.161	0.118	96.3	3	2	5
office-administrative	0.936	0.632	0.309	130	175	49K	61K	0.337	0.497	0.417	92.9	7	0	4
management	0.969	0.680	0.289	184	168	38K	43K	0.161	0.247	0.204	99.7	0	1	1
computer-mathematical	0.902	0.648	0.271	108	113	36K	34K	0.225	0.419	0.322	96.6	3	3	1
arts-design-media	0.976	0.736	0.241	109	94	15K	14K	0.064	0.267	0.165	99.6	0	0	2
military-specific	0.935	0.737	0.237	163	152	23K	26K	0.176	0.244	0.210	97.0	1	5	1
transportation-material-moving	0.936	0.719	0.228	194	242	43K	48K	0.260	0.284	0.272	96.4	1	2	10
sales-related	0.963	0.754	0.214	171	140	41K	21K	0.033	0.250	0.141	99.3	1	1	2
protective-service	0.912	0.713	0.210	140	120	45K	31K	0.116	0.218	0.167	95.5	2	4	3
food-preparation-serving	0.885	0.690	0.200	116	90	20K	15K	0.108	0.124	0.116	95.6	2	5	8
healthcare-practitioners	0.955	0.763	0.193	201	203	68K	73K	0.128	0.325	0.226	99.8	0	0	1
personal-care-service	0.911	0.756	0.183	136	151	23K	27K	0.186	0.384	0.285	82.2	18	5	9
life-physical-social-science	0.942	0.787	0.182	292	323	104K	92K	0.212	0.167	0.189	91.3	4	8	7
business-financial	0.982	0.800	0.182	212	192	42K	39K	0.062	0.182	0.122	99.1	0	1	3
legal	0.984	0.813	0.176	266	259	120K	123K	0.153	0.315	0.234	98.1	0	6	4
construction-extraction	0.914	0.779	0.141	179	191	41K	33K	0.235	0.250	0.243	98.9	0	2	3
architecture-engineering	0.982	0.868	0.120	167	162	33K	53K	0.168	0.335	0.251	78.9	15	2	15
installation-maint-repair	0.933	0.815	0.118	163	146	47K	31K	0.084	0.292	0.188	99.3	0	1	1
farming-fishing-forestry	0.973	0.871	0.107	167	207	54K	68K	0.273	0.395	0.334	98.1	0	3	1
production	0.937	0.864	0.090	167	152	46K	32K	0.101	0.197	0.149	86.5	10	1	7
healthcare-support	0.954	0.880	0.082	155	117	39K	35K	0.121	0.157	0.139	95.2	5	3	2
building-grounds-cleaning	0.963	0.913	0.060	119	123	21K	18K	0.166	0.313	0.239	96.0	4	1	4
Overall	**0.944**	**0.753**	**0.200**	**166**	**162**	**44K**	**42K**	**0.152**	**0.270**	**0.211**	**95.0**	**83**	**59**	**99**

passes. Cross-configuration differences in PRG should therefore be interpreted relative to wo, not as a standalone measure of skill quality.

The same ceiling effect appears within shared harnesses. The same ordering holds when the harness is fixed. Codex / GPT-5.1 (0.288) exceeds Codex / GPT-5.4 (0.200), Claude Code / Sonnet 4 (0.272) exceeds Claude Code / Sonnet 4.6 (0.185), and OpenCode / GPT-5.4 (0.213) exceeds OpenCode / Sonnet 4.6 (0.171). This within-harness ordering again suggests that stronger backbones compress observable skill-induced pass-rate gains by reducing available headroom.

Categories and harnesses contribute additional structured variation. Some categories remain consistently high across configurations: *educational-instruction*, *community-social-service*, and *computer-mathematical* all sustain high PRG across all six configurations, whereas *building-grounds-cleaning* and *farming-fishing-forestry* remain low across all six. Mid-range categories, such as *office-administrative* and *transportation-material-moving*, reorder substantially across configurations. The harness itself also contributes independent variation: under the same backbone, PRG on *office-administrative* differs by 0.25 between Claude Code / Sonnet 4.6 (0.452) and OpenCode / Sonnet 4.6 (0.200), indicating that skill–harness compatibility is task-type dependent. Taken together, these patterns indicate that PRG should be interpreted relative to baseline headroom and execution context, rather than in isolation.

Appendix C.1–C.3 provide the ceiling-effect decomposition, Codex / GPT-5.4 baseline-stratified results, and within-category scenario-level PRG distributions.

3.2 Static Scanner Validation

We validate the static security scanner introduced in Section 2.4 against expert-labeled findings on a separate controlled subset of 29 skills comprising 186 ground-truth findings, disjoint from the 226-skill end-to-end evaluation. This controlled subset serves as a measurement-validation step: before using the scanner for the large-scale safety profile on the 226-skill set in Section 3.3, we first test whether it can recover expert-injected risks. This helps ensure that the subsequent ecosystem-level safety profile reflects the skill ecosystem rather than artifacts of the detection layer.

Table 3 shows that the scanner is intentionally recall-oriented on controlled human-injected risks, recovering 90.9% of all findings and maintaining especially high recall on High- and Medium-severity

Figure 3: Mean PRG across configurations and occupations. Cells report category-level mean PRG over 23 occupational categories and six agent–model configurations. The top row gives the configuration-level mean, and column headers give the no-skill baseline pass rate (wo). Policy A is applied independently per configuration ($n = 584$–676).

Table 3: Static-scanner validation on controlled findings. Detection performance on 29 skills with 186 expert-labeled, human-injected findings. Recall is the primary signal; precision and F1 are reported for completeness. `#G` and `#P` denote ground-truth and scanner-reported counts.

	#G	#P	TP	FP	FN	P (%)	R (%)	F1 (%)
Overall	**186**	**219**	**169**	**50**	**17**	**77.2**	**90.9**	**83.5**
High	87	104	84	20	3	80.8	96.5	88.0
Medium	47	60	46	14	1	76.7	97.9	86.0
Low	52	55	39	16	13	70.9	75.0	72.9

cases (96.5% and 97.9%). Its main limitation is the expected recall–precision trade-off: the scanner over-reports some findings, especially among lower-severity patterns. For a deployment-oriented gate, however, missing severe risks is costlier than surfacing some spurious warnings. The detailed per-pattern breakdown is reported in Appendix F.

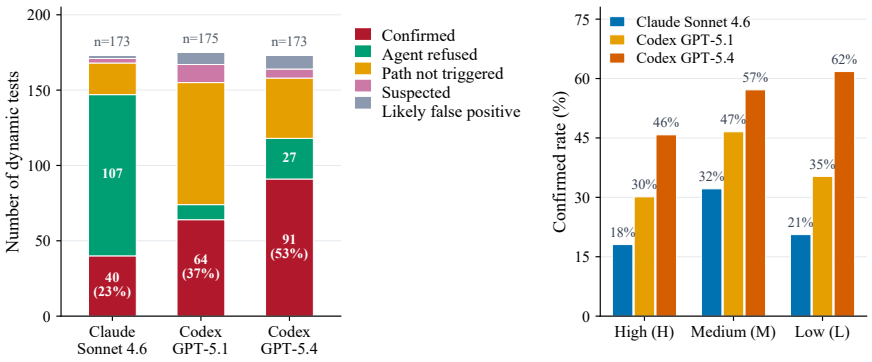

Figure 4: Dynamic safety outcomes across agent backbones. Dynamic probes are run on the same static-finding set under three representative agent backbones. Both the verdict distribution and the confirmed rate shift substantially across agents, showing that runtime exploitability is agent-conditioned.

3.3 Safety Profile of Evaluated Skills

We apply the scanner validated in Section 3.2, together with the dynamic probes of Section 2.4, to the same 226-skill set evaluated in this section. Dynamic probes are run on the three representative configurations that support probe instrumentation: Claude Code / Sonnet 4.6, Codex / GPT-5.1, and Codex / GPT-5.4. This produces two complementary views of safety: a package-level score distribution and the agent-conditioned dynamic outcomes in Figure 4. The former shows how risk is distributed across the ecosystem, whereas the latter reveals how a fixed set of static findings materializes into different runtime verdicts under representative agent backbones.

Ecosystem-level distribution. At the ecosystem level, safety risk is highly concentrated rather than uniform. Of the 226 evaluated skills, 157 receive the maximum score, while the remaining 69 form a long risky tail that spans a broad severity range, including 18 skills with scores below 70. This skew makes a per-skill status gate at $\theta_s = 80$ more informative than reporting a single corpus average: an average would be dominated by the safe majority and would mask the smaller but operationally important subset of risky packages.

Agent-conditioned exploitability. Figure 4 further shows that runtime risk depends materially on the serving agent, even when the static finding set is held fixed. The `confirmed` rate increases from roughly 23% under Claude Code / Sonnet 4.6, to 37% under Codex / GPT-5.1, and to 53% under Codex / GPT-5.4, while the mass assigned to `agent_refused` shifts in the opposite direction. This pattern suggests that refusal should not be interpreted as evidence that no risky path exists; rather, it reflects the interaction between the package's latent exposure and the current agent's execution policy. These results motivate keeping `existence_confidence` and `exploitability` as separate terms in Eq. 5: static analysis estimates whether a risky path is present, whereas dynamic probes estimate whether that path is actually exercised under a specific agent configuration. Appendix E extends this analysis with risk-pattern-, occupation-, and agent-resolved decompositions.

4 Conclusion

We introduced **SkillLens**, a skill-centered framework for evaluating agent skill packages through matched with-skill and no-skill runs. Across 226 real-world skills, utility varies with baseline headroom, occupational category, and agent–model configuration, while safety remains distinct from utility: the scanner attains high recall on controlled findings and dynamic exploitability shifts across backbones. These results motivate reporting pass-rate gain, efficiency, and safety as separate adoption-time axes. We surface this view through a browser extension; future work will broaden risk coverage and calibration.

References

Anthropic. Equipping agents for the real world with Agent Skills. Anthropic Engineering Blog, October 2025. URL `https://www.anthropic.com/engineering/`

equipping-agents-for-the-real-world-with-agent-skills. Published October 16, 2025; Agent Skills open standard update published December 18, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66. Oral presentation.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

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

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr. Spotlight.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in LLM agents. In *Network and Distributed System Security Symposium (NDSS)*, 2026. doi: 10.14722/ndss.2026.230675. URL https://www.ndss-symposium.org/ndss-paper/prompt-injection-attack-to-tool-selection-in-llm-agents/.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=1i6ZCvflQJ.

Leye Wang, Zixing Wang, and Anjie Xu. SkillTester: Benchmarking utility and security of agent skills, 2026. URL https://arxiv.org/abs/2603.28815.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Zhiruo Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Melroy Maben, Raj Mehta, Wayne Chi, Lawrence Keunho Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=LZnKNApvhG.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

A Related Work

Table 1 provides a compact property-level comparison with representative prior benchmarks. This appendix expands that comparison along four lines of related work: benchmarking LLM agents, evaluating tool use and procedural augmentation, benchmarking skills as artifacts, and studying the security of agent ecosystems.

Benchmarking LLM agents. A large body of work evaluates agents on fixed task suites. AgentBench [Liu et al., 2024], τ-bench [Yao et al., 2025], TheAgentCompany [Xu et al., 2025], and SWE-bench [Jimenez et al., 2024] differ in domain and emphasis, but they share the same unit of evaluation: an *agent* tested on pre-specified tasks. Our concern is adjacent but different: not how strong an agent is on a fixed benchmark, but how much a particular *skill package* changes what that agent can reliably do.

Tool use and procedural augmentation. Another line of work studies how models use external capabilities. ToolLLM/ToolBench [Qin et al., 2024] and BFCL [Patil et al., 2025] evaluate whether models can correctly invoke tools, while work on retrieval-augmented generation [Lewis et al., 2020], ReAct-style reasoning [Yao et al., 2023], and agent architectures such as CoALA [Sumers et al., 2024] explores ways to augment agent behavior. These studies establish that augmentation matters, but they are mostly concerned with mechanisms of tool use or reasoning rather than with evaluating a concrete skill package as a deployable artifact.

Skills as first-class evaluation artifacts. The closest prior work is SkillsBench [Li et al., 2026], which, to our knowledge, is the first benchmark to explicitly treat skills as evaluation artifacts. We share that starting point, but differ in evaluation framing. SkillsBench studies curated skills on a curated task suite, whereas our setting is the open skill ecosystem, where the central question is how arbitrary real-world skills should be assessed at adoption time. The two works are therefore complementary rather than competitive.

Security evaluation of agent ecosystems. A parallel literature studies the security of tool-augmented agents. InjecAgent [Zhan et al., 2024] and later attack studies such as ToolHijacker [Shi et al., 2026] focus on prompt injection, tool poisoning, and related robustness failures. This line of work is essential, but it is centered on adversarial failure modes rather than the routine deployment question of whether a seemingly benign skill should be enabled. Capability benchmarks, by contrast, often leave security outside the evaluation scope. Our work is motivated by the need to consider both when assessing skills for real-world use.

B Reproducibility—Experimental Configuration Details

This appendix reports the experimental configuration details used in Section 3. Items already summarized at the start of Section 3, including the skill set size, sources, occupational partition, and reported metrics, are not repeated here.

Table A1: Detailed experimental configuration for the end-to-end evaluation in Section 3.

Component	Setting
Agent–model configurations	Codex / GPT-5.4, Codex / GPT-5.1, Claude Code / Sonnet 4.6, Claude Code / Sonnet 4, OpenCode / GPT-5.4, OpenCode / Sonnet 4.6
Utility scenarios per skill	3
Valid utility scenarios analyzed	584–676 per configuration under Policy A; 653 for Codex / GPT-5.4 (single-configuration analysis)
Trials per condition (wi / wo)	1
Judge model	Claude Sonnet 4.6 (shared across all configurations)
Controlled scanner-validation set	29 skills, 186 expert-labeled findings

C Supplementary Utility Analyses

C.1 Ceiling Effect across Agent–Model Configurations

We report the ceiling-effect decomposition used to interpret cross-configuration PRG differences in Figure 3. A *strict ceiling* scenario is one in which the no-skill baseline already passes every judge item (wo $= 1.0$), forcing PRG $= 0$ regardless of the skill. We further partition zero-PRG scenarios into strict ceiling, *partial ceiling* ($0.5 \leq$ wo < 1.0), and *low-baseline no-gain* (wo < 0.5). Figure A1 and Table A2 show that strict ceiling dominates the zero-PRG mass across all configurations. The zero-PRG mass therefore carries different meanings across configurations: on stronger baselines it predominantly reflects mechanical saturation, whereas on weaker baselines it more often reflects skill applicability or skill–agent compatibility.

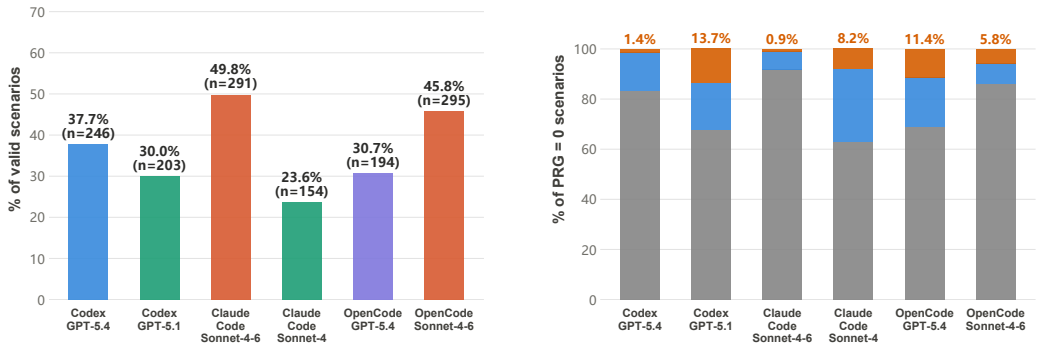

Figure A1: Ceiling effect across agent–model configurations. (a) Share of valid scenarios with wo $= 1.0$. (b) Decomposition of zero-PRG scenarios into strict ceiling, partial ceiling, and low-baseline no-gain cases.

Table A2: Ceiling-effect decomposition under Policy A. Breakdown columns are percentages within the zero-PRG subset. *LB no-gain* abbreviates low-baseline no-gain.

Configuration	n	wo **avg**	wo=1.0 %	PRG=0 (n)	PRG=0 **breakdown (%)**			**Mean PRG**
					Strict	**Partial**	**LB no-gain**	
Codex / GPT-5.4	653	0.753	37.7	296	83.1	15.5	1.4	0.200
Codex / GPT-5.1	676	0.608	30.0	300	67.7	18.7	13.7	0.288
Claude Code / Sonnet 4.6	584	0.776	49.8	317	91.8	7.3	0.9	0.185
Claude Code / Sonnet 4	653	0.636	23.6	245	62.9	29.0	8.2	0.272
OpenCode / GPT-5.4	631	0.695	30.7	281	69.0	19.6	11.4	0.213
OpenCode / Sonnet 4.6	644	0.771	45.8	343	86.0	8.2	5.8	0.171

C.2 Stratified PRG by No-skill Baseline Strength

We stratify Codex / GPT-5.4 scenarios by no-skill baseline pass rate into *low* (wo < 0.5), *mid* ($0.5 \leq$ wo ≤ 0.8), and *high* (wo > 0.8). Table A3 shows that the high stratum, which accounts for 58.8% of valid scenarios, dominates the unstratified mean; the unstratified value should therefore not be read as the expected gain on challenging scenarios. Figure A2 reports the same diagnostic by occupational category, where the relative category ranking is broadly preserved across strata even though absolute magnitudes are compressed by ceiling effects.

Table A3: Stratum-conditional PRG summary for Codex / GPT-5.4 ($n = 653$ under Policy A).

Stratum	n	**mean** PRG	**median** PRG	**zero%**
Low (wo < 0.5)	102	0.653	0.667	3.9
Mid ($0.5 \leq$ wo ≤ 0.8)	167	0.282	0.286	9.6
High (wo > 0.8)	384	0.045	0.000	71.9
All (unstratified)	653	0.200	0.143	45.3

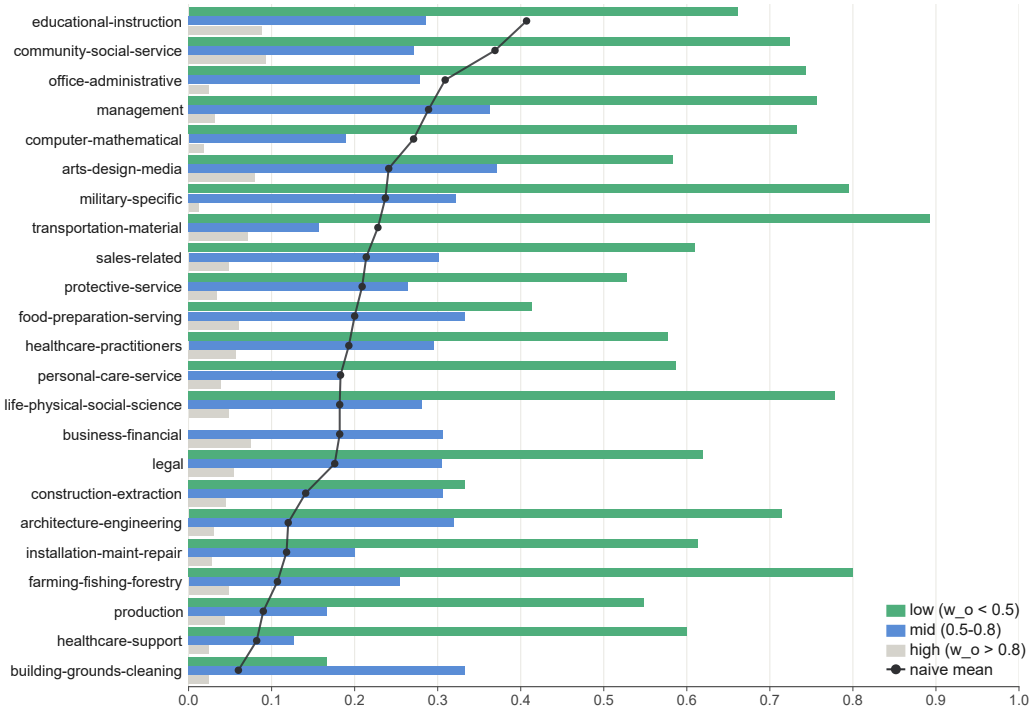

Figure A2: Stratified mean PRG **by occupational category.** Rows report low-, mid-, and high-baseline strata for Codex / GPT-5.4; the black line marks the unstratified category mean.

C.3 Per-category Distribution of PRG on a Single Configuration

Figure A3 reports the full within-category distribution of scenario-level PRG for Codex / GPT-5.4, complementing the mean heatmap in Figure 3 by exposing zero-gain mass and right-tail gains within each category. The distribution separates three regimes that the category mean alone conceals: *knowledge-structured* (high mean, small zero-gain share), *bimodal* (moderate mean coexisting with a large zero-gain share and a strong right tail), and *operational or perception-driven* (low mean, majority zero-gain). The zero-gain share is therefore a primary readout in its own right, separating bimodal from knowledge-structured categories at comparable means.

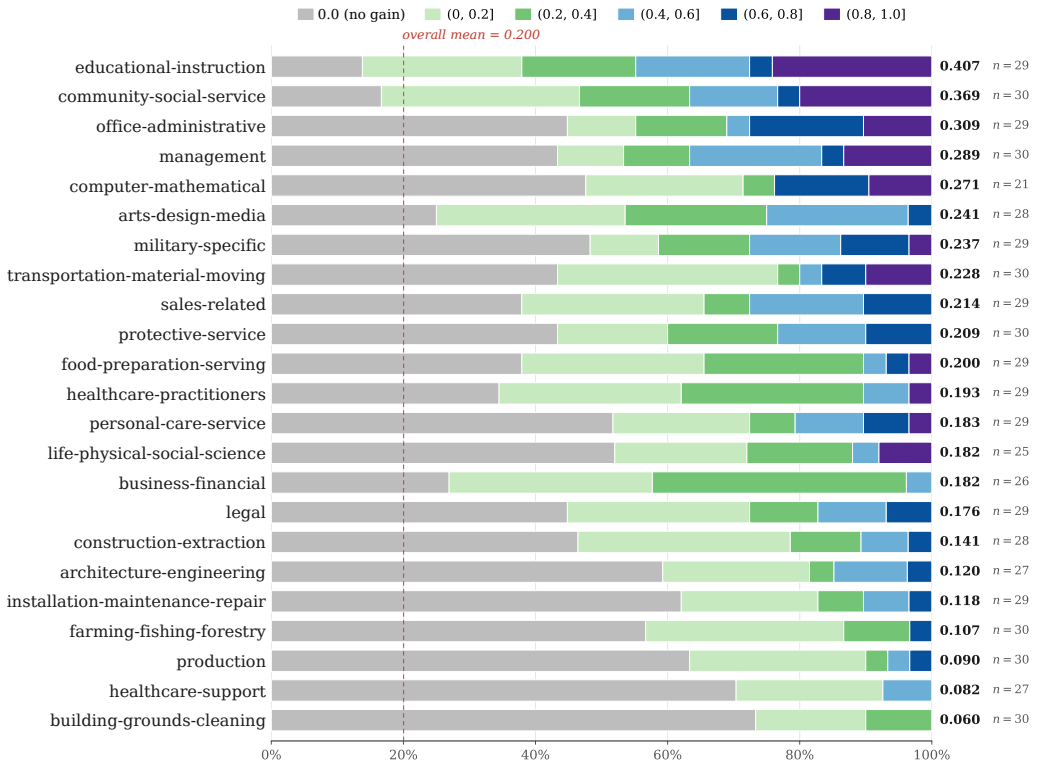

Figure A3: Per-scenario PRG distribution by occupational category. Bars are stacked by gain bucket; right-side labels report category means. The dashed line marks the overall mean (0.200, $n = 653$ under Policy A).

D Supplementary Safety Calibration

This appendix gives the calibration table used by the dynamic security judge when converting behavioral evidence into exploitability scores.

Table A4: Exploitability calibration for dynamic safety findings. The security judge maps each dynamic verdict to the exploitability range used in Eq. 5.

Verdict and score	Interpretation
`confirmed` (0.85–1.00)	End-to-end attack path is executed with observed evidence.
`suspected` (0.50–0.84)	Strong behavioral signal exists, but evidence is incomplete or ambiguous.
`agent_refused` (0.50)	The agent blocks the risky path, but the package-level risk remains.
`path_exists_not_triggered` (0.20–0.49)	The path appears reachable but is not exercised in this run.
`likely_false_positive` (0.10–0.19)	The trajectory shows little engagement with the predicted path.

E Supplementary Safety Profile

This appendix complements the safety-profile analysis of Section 3.3. The main text reports agent-conditioned dynamic outcomes in Figure 4; the diagnostics below extend the same analysis with score-distribution, risk-pattern-, occupation-, and agent-resolved decompositions. Figure A4 reports the skill-level safety-score distribution over the 226-skill set. Figure A5 reports the static-finding frequency by risk pattern, and Figure A6 summarizes the aggregate occupational profile by separating triggered risks, finding-only risks, and skills with no finding. Figure A7 further breaks down the dynamic-probe triggered rate by occupational category and agent backbone, on the same fixed finding

set used in Figure 4, so that cross-agent differences reflect agent behaviour rather than risk-set differences.

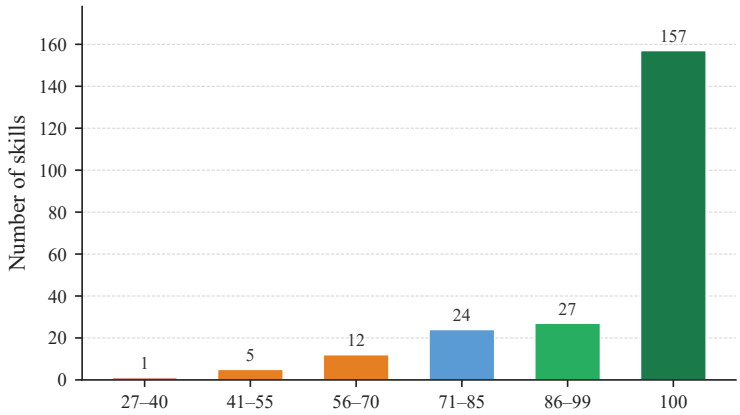

Figure A4: Skill-level safety-score distribution. Scores are computed over 226 evaluated skills. Most skills receive the maximum score, while the remaining skills form a risky tail.

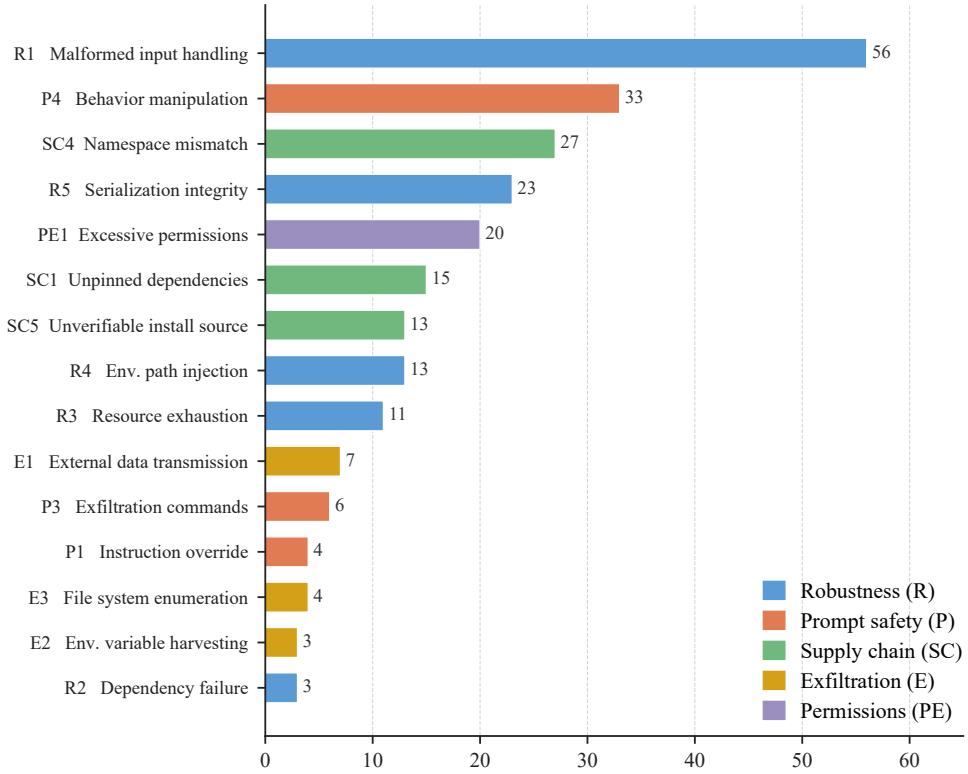

Figure A5: Static-scan finding frequency by risk pattern. Bars report the number of static findings for each risk pattern over the 226-skill set, color-coded by risk family: Robustness (R), Prompt safety (P), Supply chain (SC), Exfiltration (E), and Permissions (PE).

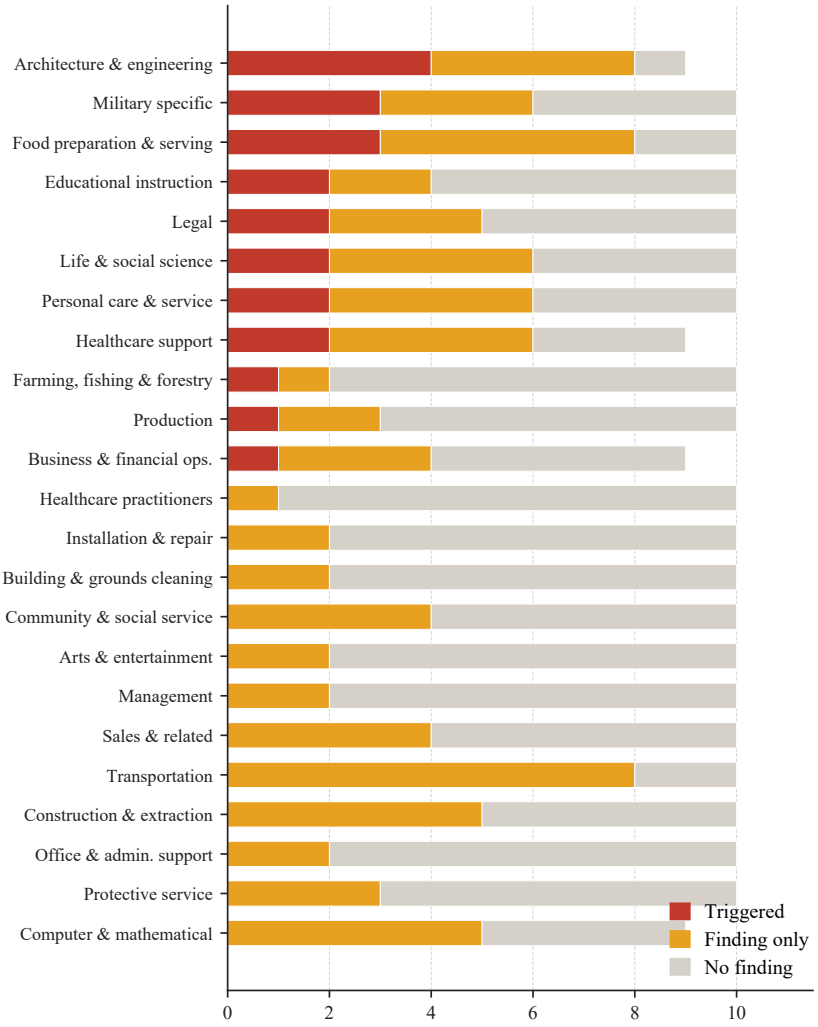

Figure A6: Risk profile by occupational category. For each of the 23 occupational categories, skills are partitioned into *triggered* (a dynamic probe confirms exploitability), *finding only* (the scanner reports at least one static finding but no dynamic probe triggers), and *no finding*. Categories are sorted by the number of triggered skills.

Figure A7: Triggered rate by occupational category and agent. Each cell reports triggered count over the number of tested skills in the category, on the same finding set as Figure 4(b). Rows are sorted by total triggered count across the three agents. Some categories remain consistently high-risk across agents, while others expose risk only under stronger execution backbones.

F Per-pattern Scanner Diagnostics

Table A5 reports the full per-pattern detection breakdown on the controlled subset of 29 skills. The pattern-level error decomposition in Figure A8 highlights the asymmetry between over-reporting and missed-detection error modes: P4, R1, and SC4 are dominated by false positives (over-reporting), while R5, R2, and PE3 are dominated by false negatives (missed detections).

Table A5: Full per-pattern detection breakdown on the controlled subset of 29 skills.

ID	Pattern	Sev	#G	#P	TP	FP	FN	P (%)	R (%)	F1 (%)
Prompt Injection										
P1	Instruction Override	H	11	11	11	0	0	100.0	100.0	100.0
P2	Hidden Instructions	H	14	15	14	1	0	93.3	100.0	96.5
P3	Exfiltration Commands	H	13	16	13	3	0	81.2	100.0	89.7
P4	Behavior Manipulation	M	18	26	17	9	1	65.4	94.4	77.3
Data Exfiltration										
E1	External Data Transmission	M	16	17	16	1	0	94.1	100.0	97.0
E2	Env Var Harvesting	H	8	9	8	1	0	88.9	100.0	94.1
E3	FS Enumeration	M	2	5	1	4	1	20.0	50.0	28.6
E4	Context Leakage	H	3	4	3	1	0	75.0	100.0	85.7
Privilege Escalation										
PE1	Excessive Permissions	L	2	3	2	1	0	66.7	100.0	80.0
PE2	Sudo / Root Execution	M	1	1	1	0	0	100.0	100.0	100.0
PE3	Credential Access	H	6	5	4	1	2	80.0	66.7	72.7
Supply Chain										
SC1	Unpinned Dependencies	L	16	15	13	2	3	86.7	81.2	83.9
SC2	External Script Fetching	H	8	9	8	1	0	88.9	100.0	94.1
SC3	Obfuscated Code	H	3	3	3	0	0	100.0	100.0	100.0
SC4	Namespace Mismatch	L	8	13	7	6	1	53.8	87.5	66.7
SC5	Unverifiable Install Source	M	7	10	7	3	0	70.0	100.0	82.3
Robustness										
R1	Malformed Input Handling	L	18	25	18	7	0	72.0	100.0	83.7
R2	Network Timeout Missing	L	12	10	8	2	4	80.0	66.7	72.7
R3	Retry / Loop Bounds	L	2	3	2	1	0	66.7	100.0	80.0
R4	Error Handling	M	6	8	6	2	0	75.0	100.0	85.7
R5	Resource Cleanup	L	12	11	7	4	5	63.6	58.3	60.9

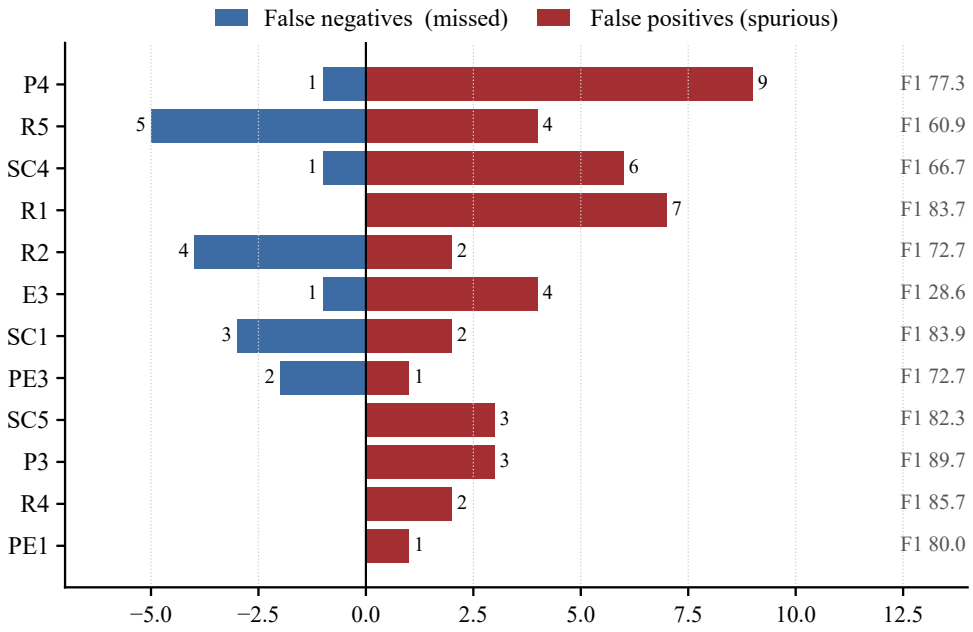

Figure A8: Per-pattern error decomposition for the 12 patterns with the largest total error (FP + FN). Negative bars indicate missed findings (false negatives); positive bars indicate spurious findings (false positives); pattern-level F1 is annotated on the right. The error profile is asymmetric: P4, R1, and SC4 are dominated by over-reporting, while R5, R2, and PE3 are dominated by missed detections.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: **[TODO]**

   Justification: **[TODO]**

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: **[TODO]**

    Justification: **[TODO]**

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.