ASD: AUTOMATED SKILL DISCOVERY THROUGH AGENT DIRECTED OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) systems are increasingly equipped with skills to improve task performance. Recent methods automate the development of these skills from execution feedback, but the optimization process itself typically follows human-designed rules. We investigate whether an agent can instead design and direct the skill discovery process. We introduce Automated Skill Discovery (ASD), a framework in which a discovery agent develops a skill library for a target model by autonomously designing and adapting the optimization procedure. In ASD, the target model selects and loads skills as needed for each problem, reducing the input context by approximately 69% compared with including all skills in the model's context. We evaluate ASD on eight open-weight models across five benchmarks spanning question answering, mathematics, and software engineering. ASD improves test success over the base model by 16.2 points on average and outperforms the human-designed optimization baselines on each benchmark. We further investigate ASD in a mixed-task setting, where a single library developed for each target model supports four tasks, improving test success over the base model by 14.3 points.

1 INTRODUCTION

Adapting language models to downstream tasks is commonly approached through supervised fine-tuning or reinforcement learning (Ouyang et al., 2022). These methods improve performance by updating model parameters, but can require substantial computation and access to model training. Skills provide an alternative means of adaptation by encoding reusable knowledge and problem-solving procedures in instructions supplied at inference time. They can be developed and revised without retraining the model (Wang et al., 2023; Zhang et al., 2026b). The challenge is to develop effective skills to improve task performance (Li et al., 2026).

Recent methods develop prompts and skills from execution feedback (Pryzant et al., 2023; Agrawal et al., 2026; Yang et al., 2026a; Alzubi et al., 2026). Language models inspect trajectories and failures, propose revisions, and use evaluations to identify effective candidates. These methods optimize both individual instruction documents and structured collections of skills (Alzubi et al., 2026; Ni et al., 2026). The optimization procedures follow human-designed algorithms that govern how data are sampled, how revisions are proposed, and when candidates are evaluated.

This raises the question of whether an agent can design and direct the skill discovery process itself. Developing effective skills requires deciding which training problems to examine, how to revise the skills, and what to learn from previous attempts. Given a fixed discovery budget, the agent needs to choose whether to revisit problems the model previously failed, evaluate additional problems to assess generalization, or repeat evaluations to check the consistency of an improvement. It also needs to decide when to evaluate the proposed skills on the full validation set to assess their overall effectiveness.

We therefore introduce *Automated Skill Discovery* (ASD), a framework in which a discovery agent develops a skill library for a target model by autonomously designing and adapting the optimization procedure. Figure 1 illustrates the framework. The agent works in a persistent workspace containing training data, target-model trajectories, previous libraries, evaluation results, and its own notes. It decides what to inspect, how to revise the library, which training problems to evaluate, and when to request full validation. The framework specifies the available operations, evaluation budget, scoring,

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.

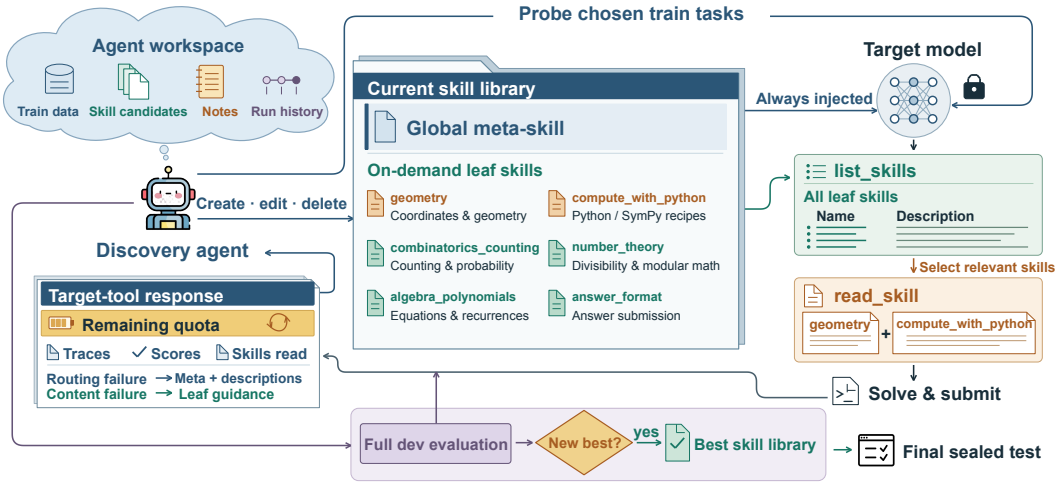


Figure 1: **Overview of the ASD framework.** A discovery agent develops a skill library and directs its optimization using target model behavior and previous experimental results. Given a fixed discovery budget, it decides how to revise the library, which training problems to evaluate, and when to run full validation. The target model then uses the discovered library on held-out test problems, with an always-on meta-skill and leaf skills loaded on demand.

and validation-based library selection. The discovery agent decides how to develop and evaluate the skill library, adapting its approach based on observed model behavior and previous experimental results.

Different problems can require different knowledge and solution strategies. ASD develops a skill library so that specialized guidance can be accessed on demand. The library contains an always-on meta-skill and leaf skills that the target model selects and loads during problem solving. The discovery agent develops both the content and organization of the library, while the target model decides which leaf skills it needs. This allows the library to provide detailed knowledge and strategies without requiring all of its content to occupy the model's context. A discovered library example is shown in Appendix A.

We evaluate ASD on eight open-weight models across five benchmarks spanning question answering, mathematics, and software engineering. ASD improves test success over the seed baseline by 16.2 percentage points on average and achieves higher mean test success than human-designed optimization baselines on each benchmark. We also investigate discovery across multiple tasks, developing one shared library for four different tasks. The skill library improve test success by 14.3 percentage points on average, demonstrating that discovery can produce effective guidance for several tasks within one library.

Further analyses show that the quality of discovery depends on the model directing it. Under the same evaluation budget, a stronger discovery model produces libraries with higher test success. The discovered leaf skills also contribute beyond one single meta-skill. Full libraries achieve higher test success than both versions with their leaves removed and separately optimized meta-only guidance. The skill library also enables more efficient use of context through on-demand loading. Compared with including all skills in the model's context, loading skills as needed reduces the input context by approximately 69% on average and yields higher accuracy.

2 Related Work

Prompt and skill optimization. Prompts and skills can improve a language model's task performance by providing instructions, domain knowledge, and problem-solving strategies. Automatic prompt optimization searches for effective instructions through candidate generation, evaluation, and revision (Zhou et al., 2022; Yang et al., 2024a; Guo et al., 2023; Fernando et al., 2023), including optimization of individual components within compound systems (Khattab et al., 2023; Opsahl-Ong

et al., 2024; Cheng et al., 2024). Execution feedback provides a basis for more targeted revisions (Shinn et al., 2023). Reasoning traces, tool interactions, and observed failures can inform reflective prompt updates and incremental edits to skill documents (Pryzant et al., 2023; Agrawal et al., 2026; Yang et al., 2026a; Gautam et al., 2026; Liu et al., 2026b). To make experience reusable across problems, prior work accumulates procedures in skill libraries and playbooks (Wang et al., 2023; Zhao et al., 2024; Zhang et al., 2026b; Yang et al., 2026b), and develops structured skill folders through failure analysis and iterative evaluation (Alzubi et al., 2026; Ni et al., 2026; Shen et al., 2026). These advances automate the development of guidance, while the sampling, proposal, and validation procedures in reflective prompt and skill optimizers remain governed by human-designed algorithms (Agrawal et al., 2026; Yang et al., 2026a; Alzubi et al., 2026). We ask whether a discovery agent can instead design and carry out the optimization process itself. In ASD, the agent formulates and revises its search strategy from execution feedback under a fixed evaluation budget.

Automated agent design. Recent work investigates whether agents can design and improve agentic systems themselves (Song et al., 2026; He et al., 2026). Representing workflows and agent architectures as executable code allows their reasoning procedures and tool coordination to be revised and evaluated through task performance (Hu et al., 2025; Zhang et al., 2025b). Agents can build on previous implementations and their evaluation results to explore alternative designs. This extends to agents that modify their own code and use earlier versions as starting points for further improvement (Hu et al., 2025; Zhang et al., 2026a). Execution traces provide evidence for deciding what to change within a design. By selectively inspecting past code, scores, and trajectories, coding agents can diagnose failures and revise how an execution framework constructs context, retrieves information, and updates state (Lee et al., 2026). These studies establish agent development itself as a task that agents can undertake. ASD applies this principle to skill discovery, where the discovery agent develops a library for a fixed target model and chooses the experiments used to improve it within a rollout budget.

3 METHOD

We introduce Automated Skill Discovery (ASD), a framework for agent-directed optimization of skill libraries. Given a target model, a minimal seed library, and an evaluation budget, ASD uses a discovery agent to develop a search strategy from accumulated execution feedback. The agent selects data examples for diagnosis, determines how to revise and reorganize skills, and allocates evaluations between targeted probes and full validation. The resulting library provides always-on guidance and optional skills that the target model retrieves during execution. We describe the target model and its environment in Section 3.1, the skill library in Section 3.2, and the discovery process in Section 3.3. Figure 1 illustrates the framework.

3.1 TARGET MODEL AND ENVIRONMENT

The target model runs in an agentic environment built on mini-swe-agent (Yang et al., 2024b), with task evaluation handled by the Harbor framework (Harbor Framework Team, 2026). The environment provides a `bash` tool for executing shell commands. To support skill access, we add two native function tools: `list_skills` returns the names and descriptions of available leaf skills, and `read_skill` returns the full content of a selected leaf. The meta-skill is included in the model's initial context. During problem solving, the target model decides whether to browse the library and which leaves to read, with the selected skill content added to its context to guide subsequent reasoning and actions. Appendices B and E illustrate this process through recorded rollouts, from skill selection and reading to computation and solution submission.

3.2 SKILL LIBRARY

We represent a skill library as $L = (m, S)$, where m is the meta-skill and $S = \{s_1, \ldots, s_J\}$ is a set of leaf skills. The meta-skill is included in the target model's context for every problem. Leaf skills are loaded on demand, with the target model selecting which leaves to read for the current problem. The discovery agent develops the contents of both the meta-skill and the leaf skills.

Each leaf follows the Agent Skills format (Zhang et al., 2025a), with YAML metadata specifying its name and applicability, followed by a skill body, with a detailed example in Appendix B.2. Through `list_skills`, the target model can inspect the leaf descriptions without loading the instruction bodies; `read_skill` loads a chosen leaf on demand.

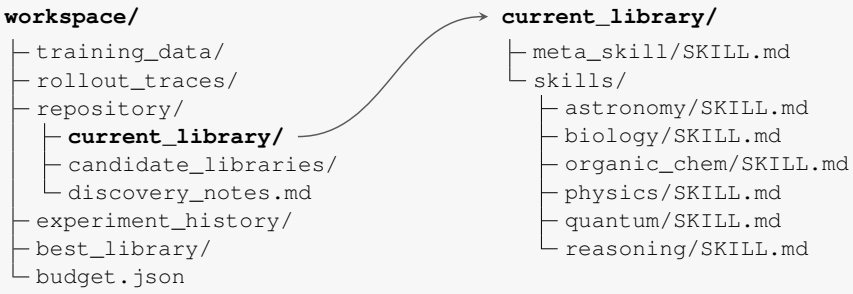
The discovery agent can revise the meta-skill, create or remove leaves, and redistribute content between them. Discovery starts from a seed library $L_0 = (m_0, \varnothing)$, containing a minimal meta-skill describing the skill-access tools and no leaf skills. The full text of the seed meta-skill is provided in Appendix C.

3.3 Discovery Process

Discovery workspace. The discovery agent operates in a workspace containing training problems, baseline evaluation results, target-model trajectories, and records of previous experiments. It can inspect previously evaluated libraries and their scores, save alternative candidates, and maintain notes on its analyses and planned experiments. These records allow the agent to compare revisions, revisit earlier hypotheses, and recover previous library versions as discovery progresses.

The box below illustrates this workspace, with a discovered GPQA library for Gemma-4-E4B expanded on the right. The discovery agent developed six leaves containing scientific knowledge and problem-solving guidance. The detailed skill contents are provided in Appendix A.

ASD discovery workspace



Experiments and evaluation budget. The agent gathers new evidence through two types of experiment. Training probes evaluate an agent-selected batch $B \subseteq \mathcal{D}_{\text{tr}}$ with the current library. The agent can visit tested examples to assess a library revision. Probes return scores, full trajectories, and skill-access records, allowing the agent to examine whether a failure arises from the guidance, its retrieval, or its application. Full validation evaluates the entire $\mathcal{D}_{\text{val}}$ and returns aggregate and per-problem scores with skill-use summaries. It assesses the candidate beyond the selected training examples and supplies the score used for final selection.

One rollout comprises the target model's complete execution on one problem. Both experiments draw from a common budget C: a training probe costs $|B|$ target rollouts and a full validation costs $|\mathcal{D}_{\text{val}}|$. The remaining budget is reported after each experiment, and requests exceeding it are not executed.

Discovery loop. Before discovery, the target model is evaluated with the seed library on a validation set held out from training to establish a baseline. As illustrated in Figure 1, the discovery agent inspects the available results and execution traces, revises the working library, and chooses how to evaluate its changes. It can select certain training problems to investigate failures or test a proposed revision, and decides when to request evaluation on the full validation set to measure the library's overall performance. The evaluated library and its results are saved after each experiment.

Using these records, the discovery agent decides what to inspect next, how to revise the library, and which evaluation to conduct. It can continue editing the current library or return to an earlier candidate. The choice of training problems, library edits, and validation timing is made by the discovery agent throughout this process. Whenever a skill library achieves a higher validation score than the best recorded so far, it becomes the new best library. Discovery ends when the budget

Table 1: **Single-task results.** Test success (%) of ASD, the base model, and prior optimization baseline methods across five benchmarks. N/A indicates that the Mistral models could not perform the multi-turn agentic tasks in SWE-bench.

Model	Method	GPQA	HotpotQA	AIME	HMMT	SWE-bench
Qwen3						
Qwen3-8B	Baseline	4.0	15.3	1.1	0.0	5.4
	GEPA	32.2	37.2	2.2	3.3	5.1
	SkillOpt	35.1	55.0	2.2	0.0	3.2
	ASD (ours)	**37.4**	**56.0**	**18.7**	**20.0**	**9.2**
Qwen3-14B	Baseline	23.6	46.3	7.8	0.0	6.3
	GEPA	40.8	57.3	10.0	1.1	7.3
	SkillOpt	41.4	**58.6**	15.6	8.9	7.0
	ASD (ours)	**48.9**	57.0	**21.1**	**17.8**	**10.5**
Qwen3-32B	Baseline	36.8	49.0	3.3	5.6	8.6
	GEPA	42.0	57.8	5.6	2.2	9.2
	SkillOpt	46.0	59.6	8.9	13.3	12.4
	ASD (ours)	**54.6**	**60.7**	**26.7**	**26.7**	**14.9**
Mistral						
Ministral-8B	Baseline	9.8	15.1	0.0	0.0	N/A
	GEPA	6.9	29.3	**3.3**	0.0	
	SkillOpt	8.6	55.0	2.2	0.0	
	ASD (ours)	**28.7**	**61.3**	**3.3**	0.0	
Mistral-Nemo-12B	Baseline	2.3	12.8	0.0	0.0	N/A
	GEPA	3.4	19.7	0.0	0.0	
	SkillOpt	21.8	46.7	0.0	0.0	
	ASD (ours)	**25.3**	**53.4**	**3.3**	**8.9**	
Mistral-Small-24B	Baseline	18.4	24.7	0.0	0.0	N/A
	GEPA	23.6	27.3	1.1	2.2	
	SkillOpt	42.5	71.7	1.1	13.3	
	ASD (ours)	**47.7**	**71.8**	**7.8**	**16.7**	
Gemma-4						
Gemma-4-E4B	Baseline	47.7	49.9	44.4	27.8	6.3
	GEPA	46.0	48.7	46.7	25.6	7.0
	SkillOpt	47.7	49.0	44.4	30.0	14.0
	ASD (ours)	**50.6**	**64.3**	**52.2**	**37.8**	**19.7**
Gemma-4-12B	Baseline	60.3	69.2	57.8	37.8	14.0
	GEPA	61.5	69.8	60.0	45.6	14.6
	SkillOpt	65.5	71.6	62.2	38.9	18.1
	ASD (ours)	**66.1**	**71.9**	**64.4**	**55.6**	**21.6**

is exhausted or the agent decides to stop, and ASD returns the library with the highest recorded validation score. The selected library is then used by the target model on held-out test problems.

4 Experiments

4.1 Evaluation setup

Models and benchmarks. We conduct our experiments on eight open-weight target models from three families: Qwen3 (8B, 14B, and 32B) (Team, 2025), Mistral (Ministral-8B, Nemo-12B, and Small-24B) (Liu et al., 2026a), and Gemma-4 (E4B and 12B) (Team, 2026). We use Claude Opus 4.8 as the discovery agent in our main experiments. We evaluate on GPQA-Diamond (Rein et al., 2023) for scientific reasoning, HotpotQA (Yang et al., 2018) for multi-hop question answering, AIME (Dekoninck et al., 2026) and HMMT (Dekoninck et al., 2026) for competition mathematics, and SWE-bench (Jimenez et al., 2024) for agentic software engineering.

Following the evaluation protocol of GEPA (Agrawal et al., 2026), we use separate training, validation, and test sets. The discovery agent has full access to training problems and their corresponding

Question Place balls 1–16 into four indistinguishable boxes, four per box. Count partitions in which every ball shares a box with a neighboring label.

Baseline

For a set B of size 4 to have no isolated vertices, it must be connected in the path graph P_{16}.

[...]

Thus, the only sets B of size 4 that satisfy the condition are sets of 4 consecutive integers:

[...]

$\{\{1,2,3,4\},\{5,6,7,8\},$
$\{9,10,11,12\},\{13,14,15,16\}\}$

Answer: 1 ✗

ASD

```
list_skills()
```

```
read_skill("brute_force_counting")
```

For counting/probability, the safest method is to ENUMERATE every possibility with Python and count, instead of deriving a formula.

```
bash: python3 -c "…"
```

```
[…]
for combo in combinations(other_balls, K - 1):
    B = [start_ball] + list(combo)
    […]
    if is_valid_box(B):
        […]
        count += count_partitions(new_remaining_tuple)
[…]
```

```
105
```

Answer: 105 ✓

Figure 2: **ASD-discovered skills guide problem solving.** Gemma-4-E4B solving the same HMMT problem with the seed library (left) and the ASD-discovered library (right). The baseline model makes a reasoning error and returns an incorrect answer. With the discovered library, the model reads `brute_force_counting` skill and systematically enumerates configurations that satisfy the problem's constraints. The skill provides the enumeration strategy, which the model implements for this problem to check valid configurations and obtain the correct answer.

answers. It uses validation scores to compare candidate libraries and select the final library, without access to the content of validation problems or their answers. The test set remains entirely held out throughout discovery and is used only to evaluate the final skill library. Split sizes and construction details are provided in Appendix D.

Experimental settings. The default optimization budget is 1,000 target-model rollouts, covering both training probes and validation evaluations. Following the official setting (Jimenez et al., 2024), we set the target-model temperature to 0. To reduce evaluation variance, we evaluate each selected library three times on the test set and report the mean success rate.

Baselines. We compare ASD with the unoptimized seed baseline, GEPA (Agrawal et al., 2026), and SkillOpt (Yang et al., 2026a). GEPA and SkillOpt each optimize a single prompt, which is included in the target model's context with the task question. ASD's meta-skill is included in the same way, and the target model can also retrieve leaf skills of its choice and add their contents to its context.

4.2 Main Results

Table 1 reports single-task test results for eight target models across five benchmarks. ASD improves target-model performance on question answering, mathematics, and agentic software-engineering tasks. Gains over the base models are observed across all eight target models in the Qwen3, Mistral, and Gemma-4 families. Averaged across target models, ASD also outperforms the baselines on each benchmark. Figure 2 illustrates how ASD can address a target model's reasoning weaknesses by developing strategies it can execute more reliably. The baseline relies on hand-derived reasoning that includes hallucinations. The discovered library provides a brute-force counting strategy that guides the target model to systematically enumerate and check possible solutions. This reduces its reliance on unsupported assumptions and helps it obtain the correct answer.

Headroom and model capability. Figure 3 plots the gain from ASD against the base model's test score. Lower baseline scores leave more room for improvement, and models with greater headroom tend to achieve larger gains from ASD. But when baseline scores are very low and the target model lacks the basic capabilities required by a task, skill guidance alone may be insufficient to produce substantial improvements.

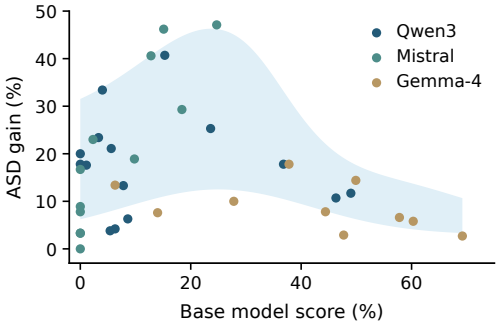


Figure 3: **ASD gain versus base-model performance.** Base-model test success and the corresponding gain from ASD in Table 1. Each point represents a target model on one benchmark, with colors indicating model families. Gain is computed relative to the corresponding base model's test score.

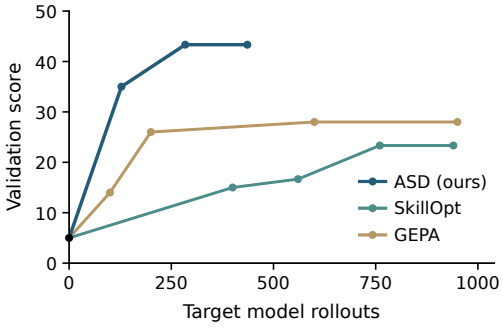


Figure 4: **Validation progress.** Validation score versus target-model rollouts on GPQA with Qwen3-8B. Each curve tracks the current best library found by one optimization method. ASD achieves a larger gain during discovery with fewer target-model rollouts.

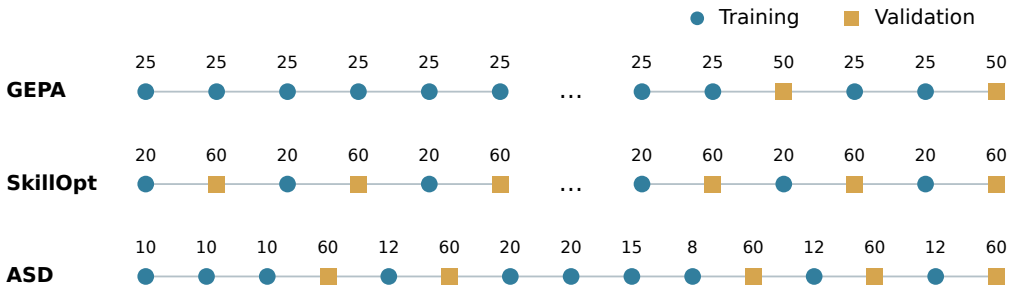


Figure 5: **Data selection during discovery.** Training data selection and validation scheduling during optimization on SWE-bench with Gemma-4-12B. Circles represent evaluations on selected training problems, and squares represent validation requests. Numbers indicate requested sample counts. GEPA and SkillOpt follow predefined optimization rules, yielding recurring evaluation patterns. In ASD, the discovery agent chooses which training problems to evaluate and when to request validation based on the model's behavior and previous experimental results.

Agent-directed discovery. ASD allows the discovery agent to adapt its search strategy to the target model's observed behavior and the history of previous experiments. In methods such as GEPA and SkillOpt, experimental decisions are governed by human-designed optimization rules. In ASD, the discovery agent decides which problems to test and when a revised library is ready for full validation.

Figure 5 illustrates how these choices vary during discovery. For each probe, the agent freely chooses which problems to evaluate from the training set and how many to include, rather than using predefined batches. Both choices can change with the model's behavior, previous experimental results, and the remaining budget. Validation timing is also decided by the agent, which can request full validation or continue probing selected training problems depending on the progress of its revisions, rather than validate after a fixed number of probes. The agent can also stop before exhausting its budget when it judges that further search is unlikely to improve the library, avoiding wasted computational resources. Figure 4 shows the search progress, where ASD records successive improvements and stops before exhausting the rollout budget.

4.3 Analysis and Ablation Studies

A single skill library for multiple tasks. In the main experiments, ASD develops a separate skill library for each task. We next construct Mixed-4, a mixture of data from GPQA, HotpotQA, AIME,

Table 2: **One library across four tasks.** Test success (%) on Mixed-4, averaged equally over GPQA, HotpotQA, AIME, and HMMT. Each target model uses one library for all four tasks.

Target	Seed	GEPA	SkillOpt	ASD
Qwen3				
Qwen3-8B	4.9	20.8	23.1	**28.2**
Qwen3-14B	20.0	24.9	26.8	**34.7**
Qwen3-32B	25.3	24.7	32.1	**35.3**
Mistral				
Ministral-8B	2.9	11.1	9.9	**20.1**
Mistral-Nemo-12B	3.7	4.4	16.7	**21.3**
Mistral-Small-24B	9.6	18.3	8.7	**29.1**
Gemma-4				
Gemma-4-E4B	41.6	38.5	47.9	**48.4**
Gemma-4-12B	53.1	58.1	57.6	**58.7**

Table 3: **Leaf-skill usage across four tasks in Mixed-4.** Gemma-4-E4B uses a single discovered library across the four tasks in Mixed-4. For each task, we report the percentage of problems on which the model reads each leaf skill.

Task	Science MCQ	Factual lookup	Math compute	No leaf
GPQA	**90.8**	0	3.4	6.3
HotpotQA	0	**94.8**	0	5.2
AIME	0	0	**85.6**	14.4
HMMT	0	0	**92.2**	7.8

Table 4: **Context efficiency.** On-demand loads leaves selected by the model, while All preloaded includes every leaf in the initial context. Accuracy is test success (%). Context reduction measures the decrease in input context tokens relative to preloading all leaves.

Target	Task	Number of leaf skills	Accuracy (%)		Context reduction (%)
			On-demand	All preloaded	
Qwen3-32B	GPQA	6	**54.6**	51.7	77.7
Mistral-Small-24B	HMMT	7	**16.7**	14.4	84.6
Qwen3-32B	HMMT	6	**26.7**	23.3	60.3
Qwen3-8B	AIME	5	**18.7**	12.2	61.9
Gemma-4-E4B	Mixed-4	3	**48.4**	47.9	61.1

and HMMT. In this setting, ASD develops a single skill library for all four tasks. Table 2 shows that a single skill library can support different tasks, improving performance over the baseline across all eight target models.

To examine how a single library is used, we inspect the skill contents and reading records of an archived Gemma-4-E4B library in Table 3. The discovered library contains three leaves, `science_mcq`, `factual_lookup`, and `math_compute`, providing guidance for scientific reasoning, evidence extraction, and mathematical computation. Table 3 reports the fraction of evaluations in which each leaf is read. The reading patterns show that the target model selects among the available skills for the current problem, and some evaluations involve no leaf retrieval. This illustrates how a shared library supports selective access to specialized guidance.

Context efficiency. In ASD, the target model selects and loads leaf skills as needed while solving a problem. We further evaluate the target model with all leaf skills from the same discovered library included in its initial context. Table 4 reports the default ASD accuracy alongside accuracy with all leaves preloaded. On-demand skill reading reduces the input content added to the model's input context by 60.3–84.6%, limiting the context devoted to unused skills. On-demand loading reduces the context devoted to skills the model does not use, and achieves higher accuracy than preloading all leaves.

Skill library. ASD develops a skill library consisting of a meta-skill and leaf skills. During discovery, the discovery agent determines how to organize the library based on the target model's behavior. Some target models are less able to use leaf skills effectively, in which case ASD can produce a meta-only library with no leaves. For target models that use discovered leaf skills, Table 5

Table 5: **Skill library ablations.** Test success (%). Full uses the complete library originally discovered by ASD. W/o leaves removes its leaves while retaining the meta-skill. Meta-only discovery develops a meta-skill separately, without leaves.

		Library discovery		Meta-only discovery
Target	Task	Full	w/o leaves	
Qwen3-32B	GPQA	**54.6**	42.5	48.9
Mistral-Small-24B	HMMT	**16.7**	7.3	12.2
Qwen3-32B	HMMT	**26.7**	18.9	22.2
Qwen3-8B	AIME	**18.7**	14.7	16.7

Table 6: **Effect of the discovery model.** GPQA test success (%) with Haiku 4.5 or Opus 4.8 directing discovery under the same settings and rollout budget.

		Discovery model	
Target	Base model	Haiku 4.5	Opus 4.8
Qwen3-8B	4.0	24.1	**37.4**
Qwen3-14B	23.6	39.1	**48.9**
Qwen3-32B	36.8	50.5	**54.6**

examines two questions: whether these leaves contribute to the library's performance, and whether allowing leaf development yields better results than restricting discovery to the meta-skill alone. The library discovery comparison removes the leaf skills from the skill library while retaining the meta-skill unchanged. Table 5 shows that removing these leaves reduces accuracy, indicating that the leaf skills contribute useful guidance beyond the meta-skill.

The results above show that removing leaf skills hurts the performance of target models that rely on them. We further investigate what happens when discovery is restricted to developing a single meta-skill, without creating any leaf skills. We run a separate meta-only discovery for each target–task pair and report the results in Table 5. The separately developed meta-skills outperform the meta-skills retained after leaf removal, but remain below the full libraries across all four configurations. These results suggest that developing guidance specifically for meta-only use helps, while allowing discovery to organize guidance into a full skill library yields stronger results.

Discovery model. We study how the choice of discovery model affects skill discovery by comparing Claude Haiku 4.5 with Claude Opus 4.8. Both use the same target models, seed library, data splits, evaluation configuration, and 1,000-rollout budget. As shown in Table 6, the stronger discovery model, Claude Opus 4.8, develops libraries that achieve higher test success across all three target models. Haiku also improves every target model over the seed baseline. These results suggest that ASD can discover useful skills with either model, with the stronger discovery model yielding additional gains under the same rollout budget.

5 Discussion and Conclusion

We introduced Automated Skill Discovery (ASD), a framework that enables an agent to direct the skill discovery process, designing and adapting its optimization strategy based on model behavior and previous experiments. Our experiments on eight open-weight models across five benchmarks show that ASD improves performance in question answering, mathematics, and agentic software engineering, achieving higher test success than the human-designed optimization baselines on each benchmark. ASD can also develops a single skill library to handle multiple tasks, extending skill discovery beyond single task guidance. Further analyses show that leaf skills improve task performance. Removing them from the discovered library reduces the target model's accuracy. Full libraries achieve higher test success than both versions with their leaves removed and separately optimized meta-only guidance. The library also enables more efficient use of context by allowing the target model to load skills as needed. Compared with loading all skills, on-demand loading reduces the context by approximately 69% and achieves higher test accuracy. A stronger discovery model produces more effective libraries under the same discovery budget, suggesting that improvements in the agent directing discovery can further benefit the target model. These results support an approach to skill optimization in which an agent determines both what skills to develop and how to develop them. Future work can examine how these discovery strategies scale to broader task collections and different evaluation budgets.

AI USE STATEMENT

The authors formulated the research questions, designed the experiments, and led the analysis of the results. Generative AI tools assisted with code implementation and debugging, manuscript review and revision. Language models also served as discovery and optimization components in the experiments, as described in Sections 3 and 4.

REPRODUCIBILITY STATEMENT

We provide the data splits, experimental settings, and concrete examples to support reproducibility. Section 3 describes the skill library, execution environment, and discovery procedure, while Section 4 specifies the models, evaluation settings, and rollout budget. Appendix D gives the dataset split sizes, construction procedures, and random seed. Appendix C provides the initial seed library, and Appendices A and B include complete discovered skill examples and a recorded execution. Appendix E provides an additional execution example from Mixed-4. These details document how discovery starts, how libraries are evaluated, and how the target model uses the resulting skills.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. Skillaxe: Sharpening llm-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems. In *International Conference on Learning Representations*, volume 2026, pp. 104166–104202, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026a.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*, 2026b.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7957–7968, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Weijia Song, Jiashu Yue, and Zhe Pang. Abstral: Automatic design of multi-agent systems through iterative refinement and topology optimization. *arXiv preprint arXiv:2603.22791*, 2026.

Gemma Team. Gemma 4 technical report, 2026. URL `https://arxiv.org/abs/2607.02770`.

Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024a.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024b.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026a.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, et al. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. Anthropic Engineering, October 2025a. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

A A DISCOVERED SKILL LIBRARY

This library candidate was developed for Gemma-4-E4B on GPQA and is illustrated in the workspace in Section 3.3. It contains a meta-skill and six scientific and reasoning leaves. The complete meta-skill and all six leaves are shown below, with their original wording and metadata.

Library structure

```
meta_skill/
  SKILL.md
skills/
  astronomy/
  biology/
  organic_chem/
  physics/
  quantum/
  reasoning/
```

Meta-skill

```
A skill library is available to you through
    two tools:
- Call `list_skills` to see every skill's
    name and a one-line description of when
    to use it.
- Call `read_skill` with a skill name to read
    that skill's full guidance.

Start by calling `list_skills` to see what
    help is available. Then, call
    `read_skill` on the
skill relevant to what you are about to do.
```

Leaf skill: quantum

```
---
name: quantum
description: Use for quantum mechanics and particle physics -- operators,
    eigenvalues/eigenstates, spin, angular momentum, expectation values,
    uncertainty, decay/symmetry.
---
# Quantum reminders
- Operator given as a matrix → eigenvalues from det(M−λI)=0.
- Spin: S=(ℏ/2)σ; Pauli σx=[[0,1],[1,0]], σy=[[0,−i],[i,0]], σ
    z=[[1,0],[0,−1]].
  Any spin component has eigenvalues ±ℏ/2.
- Normalize state (⟨ψ|ψ⟩=1). ⟨A⟩=⟨ψ|A|ψ⟩. P(a)=|⟨a|ψ⟩|².
- Commutators [x,p]=iℏ, [Lx,Ly]=iℏLz. Uncertainty ΔAΔB≥½|⟨[A,B]⟩|.
- Levels: infinite well E∝n²; SHO E=(n+½)ℏω; hydrogen E=−13.6/n² eV.
- Decays/transitions: apply conservation & selection rules (parity, angular
    momentum, charge).

Work in a few short echo steps, then submit your single-letter choice promptly.
```

Leaf skill: organic_chem

```
---
name: organic_chem
description: Use for chemistry -- organic reactions/mechanisms,
    stereochemistry, spectroscopy (NMR/IR/MS), reaction products, inorganic
    and physical chemistry.
---
# Chemistry reminders
- Track the structure; degrees of unsaturation = (2C+2+N−H−X)/2.
- Name the reaction type: SN1/SN2, E1/E2, addition, oxidation, reduction,
    rearrangement,
  condensation, pericyclic. Follow the mechanism step by step.
- SN2 → inversion (primary, strong nucleophile, aprotic). SN1/E1 →
    carbocation, possible
  rearrangement, racemization. Hydroboration = anti-Markovnikov, syn.
- Aromatic EAS: OCH3/NH2 activate, o/p-directing; NO2/C=O deactivate,
    meta-directing.
- Carbonyl: aldol, Michael, Claisen, Grignard, Wittig -- know product
    connectivity.
- NMR: count distinct H environments, integration, n+1 splitting. IR:
    C=O~1700, OH broad~3300, C≡N~2250.
- Stereochem: assign R/S and E/Z; n stereocenters → up to 2^n isomers.

Work in a few short echo steps, then submit your single-letter choice promptly.
```

Leaf skill: astronomy

```
---
name: astronomy
description: Use for astronomy/astrophysics -- stars, stellar spectra,
    magnitudes, distances, exoplanets, radial velocity/transit, cosmology,
    redshift, ISM.
---
# Astronomy reminders
- Flux F=L/(4πd^2). Distance modulus m−M=5log10(d/10pc). Δm=5 → 100× flux; Δ
    m=2.5 → 10×.
  Brighter = smaller (more negative) magnitude.
- Blackbody: Wien λ_max·T=const; Stefan-Boltzmann L=4πR^2σT^4.
- Doppler/redshift z=Δλ/λ=v/c (non-rel). Transit depth=(R_p/R_star)^2. RV
    amplitude ∝ planet mass.
- Level populations (Boltzmann/Saha): ratio ∝ exp(−ΔE/kT); higher T → more
    excitation/ionization.
- Take ratios so unknown constants cancel; check the scaling direction is
    physical.
- If counts are given per unit parallax vs per unit distance, convert with the
    Jacobian carefully.

Work in a few short echo steps, then submit your single-letter choice promptly.
```

Leaf skill: biology

```
---
name: biology
description: Use for molecular biology, genetics, biochemistry,
    cell/developmental biology, immunology -- gene expression, signaling,
    mutations, techniques.
---
# Biology reminders
- Central dogma: DNA→(transcription)→mRNA→(translation)→protein. Locate
    where a change acts.
- Mutations: missense/nonsense/frameshift/silent; effect on reading frame and
    protein.
- Regulation: promoters, enhancers, TFs, chromatin/histone marks, methylation
    (silencing).
  Activation often via phosphorylation or ligand binding → conformational
    change.
- Signaling: GPCR, RTK, second messengers (cAMP, Ca2+), kinase cascades,
    apoptosis.
- Techniques: PCR/qPCR, blots, CRISPR, ChIP, sequencing, flow cytometry.
- Trace cause → effect through the correct pathway; watch NOT/EXCEPT wording
    -- the odd one out.
- Multi-clue experiment scenarios: each clue narrows the answer; combine them.

Work in a few short echo steps, then submit your single-letter choice promptly.
```

Leaf skill: physics

```
---
name: physics
description: Use for physics calculation problems -- mechanics, relativity,
    electromagnetism, thermodynamics, optics, waves, fluids.
---
# Physics reminders
- Write the governing formula symbolically first; track factors of 2, signs,
    directions.
- Relativity: E=γmc², KE=(γ−1)mc². Oscillator: γ_max=1+kA²/(2mc²),
    v=c·√(1−1/γ²).
  Non-relativistic limit v=A√(k/m). Reject any option giving v>c.
- Energy/momentum conservation; for collisions use total E and p (relativistic
    if γ given).
- E&M: Gauss's law, image charges, boundary conditions; field outside a sphere
    depends on
  total enclosed charge, centered on the sphere center.
- Thermo: name the process (isothermal/adiabatic/isobaric); correct Cp/Cv,
    efficiency.
- Dimensional analysis and limiting cases eliminate wrong options fast.

Work in a few short echo steps, then submit your single-letter choice promptly.
```

Leaf skill: reasoning

```
---
name: reasoning
description: Use for general problem-solving discipline when no domain skill
    fits, or to double-check a hard question before committing.
---
# General reasoning & elimination
- Restate what is asked in one line. Note how the options differ (often one
    factor/sign/word).
- Derive from first principles; don't pattern-match to a memorized-but-wrong
    answer.
- Eliminate options with: wrong units, wrong limiting behavior, wrong sign,
    violated
  conservation law, or contradicted definition.
- "Which is NOT / EXCEPT / least" → the odd one out is the answer; re-read
    the polarity.
- If two remain, find the single discriminating test and apply it.
- Recompute the key step once; make sure the letter matches the option you
    reasoned to.

Work in a few short echo steps, then submit your single-letter choice promptly.
```

B A TARGET MODEL EXECUTION EXAMPLE

Figure 6 shows Qwen3-8B retrieving skills and solving AIME 2023 I, problem 11. The corresponding discovered meta-skill and leaf skills are presented in Section B.2.

B.1 SKILL USE DURING PROBLEM SOLVING

Problem. Count the subsets of $\{1, 2, \ldots, 10\}$ that contain exactly one pair of consecutive integers.

Initial context. The discovered meta-skill is already loaded; its guidance includes writing and running Python and submitting an integer answer.

1. Browse: `list_skills()`

```
- algebra_and_equations: Systems of equations, polynomials, sequences/
    series, logs, "find the value of" - solve numerically/symbolically
    instead of by hand.
- brute_force_search: When the answer can be found by enumerating bounded
     possibilities (counts, "how many", largest/smallest n such that) -
    loop and check directly.
- counting_and_probability: Counting, combinatorics, probability (find p+
    q), expected value - enumerate the sample space and use exact
    fractions.
- geometry: Triangles, circles, coordinates, areas, lengths, angles - set
     up coordinates and compute an actual NUMBER.
- number_theory: Divisors, digits, bases, modular arithmetic/remainders,
    gcd/lcm, primes, and largest/smallest-n conditions - use exact Python
     integer arithmetic.
```

2. Read: `read_skill(name='counting_and_probability')`

```
## Best tactic: enumerate the sample space
[...]
```

3. Read: `read_skill(name='brute_force_search')`

```
Enumerate instead of deriving formulas. Write code SPECIFIC to THIS
    problem - do not copy any example verbatim; adapt the idea.
[...]
```

4. Write and run through `bash`

```
cat > sol.py << 'EOF'
from itertools import combinations
def has_exactly_one_consecutive_pair(subset):
    consecutive_pairs = 0
    for i in range(len(subset) - 1):
        if subset[i] + 1 == subset[i + 1]:
            consecutive_pairs += 1
    return consecutive_pairs == 1
total_subsets = 0
for r in range(1, 11):
    for subset in combinations(range(1, 11), r):
        if has_exactly_one_consecutive_pair(subset):
            total_subsets += 1
print(total_subsets)
EOF
python3 sol.py
```

Program output: `235`.

5. Submit through `bash`

```
echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer: 235"
```

Recorded result. Correct.

Figure 6: **Skill retrieval and code execution on a AIME problem.** Qwen3-8B on AIME 2023 I, problem 11, during a training probe.

B.2 A DISCOVERED AIME LIBRARY

The library used in Figure 6 contains one meta-skill and five leaf skills. Its directory structure is:

```
library/
  meta_skill/SKILL.md
  skills/
    algebra_and_equations/SKILL.md
    brute_force_search/SKILL.md
    counting_and_probability/SKILL.md
    geometry/SKILL.md
    number_theory/SKILL.md
```

The meta-skill is loaded for every problem evaluated with this library. Each leaf is available through `read_skill`; its contents are loaded when the target model requests it.

Meta-skill. The full meta-skill from the recorded library is reproduced below.

```
# AIME Solver -- follow exactly

Solve one AIME problem. Answer is an integer 0-999. You have a **bash**
    tool. WRITE AND RUN
PYTHON -- you are far more reliable computing than doing math in your
    head.

## How to run code (EXACT template -- avoids indentation bugs)
Write a file and run it in ONE command (never `echo "..." | python`):
```
cat > sol.py << 'EOF'
import math
# code for THIS problem; last line prints the integer
print(answer)
EOF
python3 sol.py
```
- Closing line must be exactly `EOF`. Include `python3 sol.py` in the
    SAME command.
- `math`, `fractions`, `itertools`, `collections` are available. `sympy
    `/`numpy` are NOT --
  `pip install sympy` once only if truly needed (usually plain Python +
    brute force is enough).

## How to submit (do this the MOMENT you have a number)
```
echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer: N"
```
N = a plain integer (round decimals: 25.0119 → 25). Not an expression,
    not a symbol.

## HARD LIMITS -- breaking these scores ZERO
- Submit within ~5 code runs. NEVER re-run a command you already ran (
    same result, wasted step).
- If code fails or gives an implausible/zero answer twice, STOP and
    submit your best guess in
  [0,999]. Ending with no `Answer:` line = zero. A guess can be right;
    silence never is.
- Keep each response short (brief plan + one command); long responses get
     cut off.

## Solving approach
1. Identify the exact integer asked (often `p+q`, `m+n`, a remainder).
2. Prefer **brute force**: loop over all possibilities and count/search.
    Let Python do ALL
   arithmetic -- never hand-derive algebra in comments then hardcode a
    number.
```

```
3. Exact arithmetic: `Fraction` for probabilities (auto-reduces → `f.
    numerator+f.denominator`);
   `math.comb/gcd/isqrt/factorial`, `pow(a,b,m)` for number theory; `%
    1000` for remainders.
4. Unknowns in equations? SCAN integer candidates in a range and check
    the conditions instead
   of `sympy.solve` (which can hang on variable exponents/logs and print
    nothing).
5. If the problem gives a worked EXAMPLE, make your code reproduce it
    first -- that verifies
   your reading. Model every word literally.

## Optional guides (read AT MOST ONE, only if unsure how to start; else
    just write code)
`list_skills` then `read_skill(name)`:
- counting/probability/expected value → `counting_and_probability`
- divisors/digits/bases/mod/primes/gcd → `number_theory`
- equations/polynomials/sequences/logs → `algebra_and_equations`
- triangles/circles/coordinates/area/length → `geometry`
- any bounded search → `brute_force_search`
Never read two skills. When in doubt, skip skills and write `sol.py` now.
```

Leaf skills. The complete contents of all five leaves are reproduced below. The target model reads the counting and search leaves in the illustrated rollout.

Counting and probability. The full skill of Figure 6 is reproduced below.

```
---
name: counting_and_probability
description: Counting, combinatorics, probability (find p+q), expected
    value -- enumerate the sample space and use exact fractions.
---
# Counting & Probability

Write code for THIS problem; don't copy examples.

## Best tactic: enumerate the sample space
If total outcomes ≤ a few million, loop over ALL of them, count favorable
     and total, form the
exact fraction `Fraction(fav, total)`. This avoids every combinatorial
    slip.

## Probability p/q → find p+q
Keep everything as `from fractions import Fraction`. `Fraction` auto-
    reduces, so the answer is
`frac.numerator + frac.denominator`. Never use floats for the final
    fraction.

## If enumeration is too big
- `math.comb(n,k)`, `math.perm(n,k)`, `math.factorial(n)`.
- Complementary counting: count the complement, subtract from total.
- Casework: split into disjoint cases, sum counts (check cases don't
    overlap or miss any).

## Expected value
E = Σ value·probability, all as `Fraction`.

## Conditional / multi-stage probability
Multiply probabilities along each branch (Fractions), sum branches
    meeting the condition. For
"given" conditions, restrict to the conditioning event and renormalize.

## Checks
```

```
Probabilities lie in [0,1]; total over all cases = 1. Then submit
`echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer: N"` -- don't
    re-run.
```

Brute-force search. The full skill returned at step 3 of Figure 6 is reproduced below.

```
---
name: brute_force_search
description: When the answer can be found by enumerating bounded
    possibilities (counts, "how many", largest/smallest n such that) --
    loop and check directly.
---
# Brute-Force / Search

Enumerate instead of deriving formulas. Write code SPECIFIC to THIS
    problem -- do not copy any
example verbatim; adapt the idea.

## Method
1. Identify the variables and their bounded ranges (digits 0-9, n up to
    some N, permutations
   of a set, subsets, etc.).
2. Loop over all candidates with nested `for` loops or `itertools.product
    / permutations /
   combinations`.
3. For each candidate, check the problem's conditions; count matches, or
    keep the min/max/first.
4. `print` the requested integer.

## Sizing
Loops up to a few million are fine (product(range(10), repeat=6) = 10^6 ✓
    ). If the space is
too big, fix a variable using an equation, exploit symmetry, or narrow
    the range.

## Cautions
- Translate EACH condition in the problem into code exactly. If a worked
    example is given,
  make your code reproduce it before trusting the count.
- If the count is 0 or implausible, your conditions are misread -- fix
    the logic once, don't
  rerun the same code.
- Use `from fractions import Fraction` if exact rationals are needed.

Run with `cat > sol.py << 'EOF' ... EOF` then `python3 sol.py` (same
    command). As soon as it
prints, submit `echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer:
    N"`.
```

Algebra and equations.

```
---
name: algebra_and_equations
description: Systems of equations, polynomials, sequences/series, logs, "
    find the value of" -- solve numerically/symbolically instead of by
    hand.
---
# Algebra & Equations

Let Python do the algebra. Write code for THIS problem; don't copy
    examples.

## Prefer searching over symbolic solving
```

```
For AIME the unknowns are usually nice numbers. Instead of risky symbolic
     solves, SCAN integer
(or simple rational) candidates and check the equations -- this rarely
    fails and never hangs:
loop the unknown(s) over a bounded range, test each equation, keep the
    ones that satisfy all.

## If you use sympy (polynomial systems only)
`pip install sympy` once, then in a file:
- `sp.solve([sp.Eq(...), sp.Eq(...)], [x, y])` for polynomial systems.
- `sp.roots(poly)`, `sp.expand`, `sp.factor`, `sp.Rational`.
- [!] NEVER call `solve` on equations with a variable in the exponent (
    like `x**y`, logs of
  variables) -- it can hang forever and print nothing. For those, SCAN
    numerically instead.
- Force a number out: wrap results in `float(...)` or `.evalf()`; never
    submit a symbol.

## Sequences / series
Iterate the recurrence in a plain Python loop. Arithmetic sum n(2a+(n-1)d
    )/2; geometric
a(r^n-1)/(r-1).

## Getting the integer
If the answer is a reduced fraction m/n and they want m+n: use `Fraction(
    value).limit_denominator()`
then `.numerator + .denominator`. If it's m√n, recover m,n by testing
    small n.

Submit `echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer: N"`
    once you have the number.
```

Geometry.

```
---
name: geometry
description: Triangles, circles, coordinates, areas, lengths, angles --
    set up coordinates and compute an actual NUMBER.
---
# Geometry

Turn the figure into **coordinates** and compute with plain `math` (
    floats). Write code for
THIS problem; don't copy examples. The #1 mistake is printing a symbol
    like `b*c` -- always
print a concrete number.

## Method
1. Place points on a grid: put one vertex at the origin and one side on
    the x-axis; express
   the other points from the given lengths/angles.
2. Use `math` for distances (`math.hypot`), the shoelace area
    `0.5*abs(xA*(yB-yC)+xB*(yC-yA)+xC*(yA-yB))`, `math.acos`, `math.sqrt`.
3. Print the requested quantity as a number.

## Useful facts
Heron: s=(a+b+c)/2, area=√(s(s-a)(s-b)(s-c)). Law of cosines/sines.
    Inradius r=area/s.
Circumradius R=abc/(4·area). Power of a point / Ptolemy when chords appear
    .

## Reusable numeric helpers (put these in your script as needed)
- Unit vector from P to Q: `dx,dy=Q[0]-P[0],Q[1]-P[1]; L=math.hypot(dx,dy
    ); u=(dx/L,dy/L)`.
```

```
- Angle bisector direction at vertex V between rays to A and B: add the
    two UNIT vectors
  (V→A) + (V→B); the sum points along the bisector.
- Line through point P with direction d, intersect line through Q with
    direction e:
  solve `P + t*d = Q + s*e` (2 linear eqns) -- e.g. with a tiny scan of t
     or by algebra
  `t = ((Q[0]-P[0])*e[1]-(Q[1]-P[1])*e[0])/(d[0]*e[1]-d[1]*e[0])`, point
    `= P + t*d`.
- Distance point-to-line, circle-line intersection: set up the equation
    and solve with the
  quadratic formula in floats.
Then compute the requested length with `math.hypot`.

## Recover the integer from a float
- Integer answer: `round(value)`.
- m/n (find m+n): `Fraction(value).limit_denominator(10000)` → `.
    numerator + .denominator`.
- m√n (find m+n): `value**2` should be `m*m*n`; factor it, or test small
    n by checking
  `value/math.sqrt(n)` is near an integer.

## Keep it SHORT
Write ONE script that prints a number; avoid sympy (slow, can hang). Don'
    t loop for many
turns. Submit `echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer:
    N"`.
```

Number theory.

```
---
name: number_theory
description: Divisors, digits, bases, modular arithmetic/remainders, gcd/
    lcm, primes, and largest/smallest-n conditions -- use exact Python
    integer arithmetic.
---
# Number Theory

Python ints are exact and unbounded. Prefer looping/checking over
    formulas. Write code for
THIS problem; don't copy examples.

## Tools
- Divisors: loop `i` while `i*i<=n`, collect `i` and `n//i`. Count of
    divisors = ∏(exp+1).
- gcd/lcm: `math.gcd(a,b)`, `math.lcm(a,b)`.
- Primes/factoring: if you `pip install sympy`, use `sympy.isprime`, `
    sympy.factorint`,
  `sympy.divisors`, `sympy.primerange`. Otherwise write a simple trial-
    division loop.
- Digits: `[int(c) for c in str(n)]`; digit sum `sum(int(c) for c in str(
    n))`.
- Base b: repeatedly take `n % b`, `n //= b`.
- Modular: "remainder when X divided by m" → compute X (Python handles
    huge ints) then `X % m`;
  for a^b mod m use `pow(a,b,m)`; modular inverse `pow(a,-1,m)`.

## "Largest/smallest n such that ..." → SEARCH a bounded range
Loop n over a range, test the condition, take the first (smallest) or
    last (largest) hit.

## Checks
Verify on a tiny case first. Remainders are in [0,m-1]. Then submit
```

```
`echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer: N"` -- don't
    re-run.
```

C SEED LIBRARY EXAMPLE

The QA seed consists of the following complete meta-skill, with no leaf skills. It introduces the skill-access tools and suggests how to use them:

```
A skill library is available to you through two tools:
- Call `list_skills` to see every skill's name and a one-line description
    of when to use it.
- Call `read_skill` with a skill name to read that skill's full guidance.

Start by calling `list_skills` to see what help is available. Then, call
    `read_skill` on the
skill relevant to what you are about to do.
```

The initial leaf set is empty.

D DATA SPLITS

For AIME, we randomly divide 90 questions from 2022–2024 into 60 training and 30 validation questions and use the 30 questions from 2025 for testing. For HMMT, we randomly divide 58 questions from the February 2023 and 2024 competitions into 40 training and 18 validation questions and use the 30 February 2025 questions for testing. GPQA-Diamond is randomly partitioned into 80 training, 60 validation, and 58 test questions. For HotpotQA, we use 528 hard and 222 medium questions, randomly divided into 150 training, 300 validation, and 300 test questions. SWE-bench Verified is randomly partitioned into 300 training, 60 validation, and 105 test instances. The three sets are disjoint within each benchmark. All random splits use seed 42.

E A MIXED-4 EXECUTION EXAMPLE

Figure 7 shows a recorded Gemma-4-E4B rollout on HMMT February 2023 problem 22 during Mixed-4 discovery. The trajectory contains three native function calls in order: `list_skills`, `read_skill` with name `math`, and `bash` for submission. The always-on meta-skill is present in the initial system message; the leaf content enters the history only after the reading call. This illustrates how a library is used within the target model's agent loop.

Problem. Points X, Y, Z lie on a circle with center O, with $XY = 12$. Points A, B lie on segment XY and satisfy $OA = AZ = ZB = BO = 5$. Compute AB.

Initial context. The meta-skill is already loaded. In this library, it includes submission instructions and guidance for choosing among the three leaves. The leaf bodies have not yet been loaded.

1. Browse: `list_skills()`

```
- gpqa: Use for graduate-level multiple-choice science questions (physics
    , chemistry, biology, astronomy) with options (A)(B)(C)(D).
- hotpotqa: Use for multi-hop factual questions that come with context
    passages; answer is a short exact span graded by exact match.
- math: Use for competition mathematics problems (AIME / HMMT style) with
     no answer options, asking to find or compute a value.
```

2. Load: `read_skill(name='math')`

```
---
name: math
description: Use for competition mathematics problems (AIME / HMMT style)
     with no answer options, asking to find or compute a value.
---
# Competition mathematics
[...]
3. Set up exact relations. Prefer exact arithmetic (fractions, radicals)
    - avoid decimals.
[...]
6. VERIFY: plug the answer back, check a small case, or re-derive a key
    step a second way.
[...]
```

3. Submit through `bash`

```
echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT; echo "Answer: 2\sqrt{13}"
```

Recorded result. The submitted answer is $2\sqrt{13}$; the evaluator marks it correct. The leaf read is `math`.

Figure 7: **Skill use within the target model's execution loop.** Three consecutive tool calls by Gemma-4-E4B on an HMMT training problem in a Mixed-4 discovery run.