Skill-Synthesizer: Query-Aware Skill Synthesis for Scientific Reasoning

Jacob McCarran [1] **Rajsuryan Singh** [1] **Carlos Arribalzaga Jové** [1] **Khaled Ahmed** [1] **Marco Del Tredici** [1]

Abstract

Maximizing the value of an agent's accumulated experience requires more than retrieving the most similar skill: it requires selectively combining and adapting knowledge based on the demands of the current problem. This is especially true in scientific reasoning, where solving a problem often requires integrating theoretical principles and procedural knowledge from multiple sources. Existing skill-based memory systems fall short of this: they retrieve whole skills and inject them directly into the agent's context, without reasoning over their content or relationships. This is suboptimal when relevant knowledge is scattered across multiple skills, or when retrieved skills contain information only partially relevant to the current task. We present **Skill-Synthesizer**, which addresses these limitations by retrieving relevant passages from a skill library and composing them into a unified, query-specific skill. A dedicated synthesis agent reasons over how retrieved passages complement each other, assembles them into an executable procedure, and verifies intermediate steps through code execution, a highly valuable capability for the derivation-heavy problems that characterize scientific workflows. Evaluated on two challenging physics benchmarks, Skill-Synthesizer consistently outperforms all retrieval baselines in both accuracy and reasoning soundness, with especially strong gains on a graduate-level benchmark where no other retrieval method improves over the memoryless baseline.

1. Introduction

Scientific reasoning requires identifying recurring patterns, reusing procedural knowledge, and transferring concepts across problems. As AI agents become increasingly integrated into scientific workflows, an important open challenge is enabling them to accumulate and effectively reuse knowledge acquired from prior experience. Recent work on self-improving agents has demonstrated that external memory systems can help agents iteratively refine algorithms, discover reusable strategies, and autonomously generate scientific artifacts (Lu et al., 2024; Toscano et al., 2026; Novikov et al., 2025; Romera-Paredes et al., 2024).

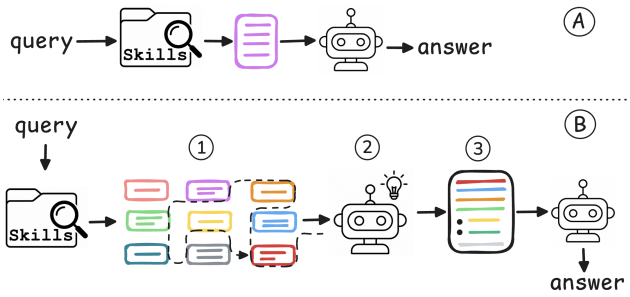

Figure 1. A: in the standard approach, a single skill (or a few skills) is retrieved and passed to the agent. B: our approach: ① relevant passages are extracted from the existing skills; ② a specialized agent reasons over them, and ③ synthesizes the information into a new, query-specific skill.

An increasingly common way to leverage prior experience is through the use of agent skills (Anthropic, 2025b;c; Wang, 2025; Xia et al., 2026). After performing a task, an agent analyzes its *trajectory* – i.e., the concatenation of the query, reasoning steps, tool calls, output, and environmental feedback – reflects over it, and produces one or more skills, or updates existing ones. Skills are usually quite extensive, as they can contain task-specific details as well as procedural guidance. In future tasks, the agent searches for the most useful skill or n most useful skills and includes them in its context.

While skills have been shown to improve performance (Xia et al., 2026; Ye et al., 2026; Zhou et al., 2026), we note that the process of simply retrieving and injecting the most relevant (n) skills into the agent's context is suboptimal, for two reasons. First, the information needed to perform the new task might be scattered among a large number of skills. This is especially true for scientific tasks, which by nature require combining theoretical principles, problem-specific

[1]Axiomatic AI. Correspondence to: Jacob McCarran <jacob.mccarran@axiomatic-ai.com>, Marco Del Tredici <marco.deltredici@axiomatic-ai.com>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

heuristics, and domain knowledge from multiple sources. One solution would be to add all relevant skills to the agent's context, but this would potentially hurt performance due to noise (Liu et al., 2024; Zhang et al., 2025a). Furthermore, as the number of relevant skills grows (Wang, 2025; Xia et al., 2026), adding all relevant skills (Hong et al., 2025) will eventually hit the maximum input length of the agents.

Second, skills are typically stored in a flat, unstructured directory, with no relationships encoded between them. However, solving a complex problem often requires synthesizing complementary pieces of information across multiple sources of knowledge, rather than relying on a single one (Li et al., 2026). Furthermore, which pieces should be combined depends on the task being performed. For example, optimizing the energy efficiency of a piston engine may require composing skills related to (a) piston motion, (b) heat exchange, and (c) friction, whereas simulating how a hot air balloon reaches equilibrium altitude may require the same (b) heat exchange skill together with (d) buoyancy, (e) pressure gradient, and (f) gas volume skills. Notably, the (b) heat exchange skill will be applied differently in each scenario: hence, even when skills required for different problems overlap, the way those skills must be combined depends on the query. This motivates the need for dynamic, query-dependent skill composition.

In this work, we address these limitations and focus on the task of how to best extract and synthesize the information from an existing skill library. We propose a novel methodology to retrieve and synthesize information from a large set of skills. Instead of pulling the best skill (or n skills) as in the standard approach (Fig. 1A), in our approach (Fig.1B), we first retrieve only the most relevant *passages*, i.e., segments of skills, and then pass them to a Skill-Synthesizer agent, that reasons over passages, verifies intermediate steps via code execution, and synthesizes a new query-specific skill.

Our approach addresses the two main limitations above. First, by retrieving only relevant passages rather than entire skills, it explores the information in the skill directory at a finer-grained level, extracting only the relevant content while discarding unnecessary or noisy information. Second, it allows the agent to synthesize query-specific insights across independently retrieved pieces of information and aggregate them into a new executable skill. Rather than treating retrieved skills as independent inputs, the Skill-Synthesizer explicitly reasons over how the retrieved fragments interact and complement each other, restructuring them into a unified solution procedure tailored to the target problem. To support this reasoning process, Skill-Synthesizer is equipped with a Python execution environment that allows it to programmatically verify intermediate steps, perform symbolic manipulation, and check the consistency of derived expressions. These capabilities are highly valuable for the derivation-heavy problems that characterize scientific workflows.

We evaluate our approach on complex physics tasks, which demand accurate reuse and composition of knowledge acquired from prior problems. We used two challenging datasets, UGPhysics (Xu et al., 2025) and PHYSICS (Feng et al., 2025), and compared our approach to other retrieval methods. The results show that Skill-Synthesizer consistently outperforms all retrieval baselines on both datasets in accuracy and reasoning quality, with particularly strong gains on the challenging PHYSICS questions, where no other approach achieves a significant improvement over the Memoryless baseline. These results suggest that the ability to filter irrelevant information and dynamically connect complementary knowledge at query time is crucial to unlocking the full potential of skill-based memory systems in scientific reasoning.

2. Related works

2.1. Prompt Optimization and Context Engineering

Agents can improve their performance over time by leveraging lessons learned from previous trajectories at test time (Shinn et al., 2023; Wang et al., 2023; Zhang et al., 2025a; Zhou et al., 2026). Reflexion (Shinn et al., 2023) generates natural language reflections from previous failures and appends them to future prompts; in a similar vein, methods like DSPy (Khattab et al., 2023), GEPA (Agrawal et al., 2025), Meta-Prompting (Suzgun & Kalai, 2024), and MIPROv2 (Opsahl-Ong et al., 2024) optimize prompts, reasoning templates, and examples through iterative refinement. Recent approaches such as Dynamic Cheatsheet (Suzgun et al., 2026) and Agentic Context Engineering (ACE) (Zhang et al., 2025a) go further by building query-specific contexts from past experiences. However, both maintain a single evolving document that is injected directly into the model's context, which becomes increasingly difficult to manage as experience accumulates — a problem ACE itself refers to as *context collapse* (Zhang et al., 2025a). We address this by distributing experience across a structured skill library, and focusing on how to identify, combine, and synthesize the most relevant pieces of information in it.

2.2. Skill-based Agentic Memory

Skill-based memory systems have recently emerged as a dominant paradigm for long-term agent learning (Wang et al., 2023; Anthropic, 2025b;c), with pipelines that store reusable lessons, workflows, and code snippets in structured formats showing consistent improvements on real-world tasks. SkillRL (Xia et al., 2026), Memento (Zhou et al., 2026), and Meta Context Engineering (Ye et al., 2026) re-

trieve relevant skills from memory and incorporate them into the model's context at test time.

These methods retrieve whole skills and inject them directly into the agent's context, without any reasoning over their content or relationships. When the information needed to solve a task is spread across multiple partially relevant skills, this approach has two shortcomings: first, relevant knowledge may be buried within a larger skill alongside irrelevant content; second, no mechanism exists to reason over how retrieved skills relate, overlap, or complement each other. Both limitations grow more severe as the skill library scales (Wang, 2025; Xia et al., 2026).

The concurrent work SkillGraph (Li et al., 2026) addresses this by embedding skills in a static graph, enabling retrieval of structured subgraphs rather than isolated skills. Our approach differs in a key way: rather than precomputing fixed relationships offline, Skill-Synthesizer dynamically synthesizes a new skill tailored to the query at hand, explicitly reasoning over how retrieved fragments interact and complement each other.

Beyond the retrieval problem, existing methods introduce practical constraints that limit applicability: SkillRL (Xia et al., 2026) requires additional fine-tuning to teach the model to use retrieved skills effectively, Memento (Zhou et al., 2026) relies on a trained retriever, and Meta Context Engineering (Ye et al., 2026) requires iterative test-time optimization loops. In contrast, Skill-Synthesizer requires none of these: it operates directly with frontier reasoning models, without retriever training, model fine-tuning, or multi-pass optimization.

3. Methodology

3.1. Skills

Skills are a form of procedural memory distilled from prior trajectories that contain task-specific guidelines and reusable solution strategies. Skills encode how to do something rather than just factual knowledge. Following recent skill-based agent memory approaches (Anthropic, 2025b; Xia et al., 2026; Zhou et al., 2026; Ye et al., 2026) we implement skills as `.md` files with a clear and informative name (e.g., *adiabatic-invariance-particle-in-box*), a top-level description, and the following sections: *When to Use*, *Core Procedure*, *Worked Example*, *Critical Decision Rules*, *Common Mistakes*, *Variants and Edge Cases*, *When NOT to Use*. An example of skill is provided in Appendix A.1.

3.2. Skill generation

In this work, we focus on the task of retrieving the best possible information from an existing set of skills. As a first step, we construct such a set. We follow a standard procedure (Zhou et al., 2026): each question in the training split of a dataset (see Section 4.1) is passed to a Solver agent, which generates an answer, that is then assigned a binary score by a Judge agent.[1] Finally, the initial query, the answer, the score, and ground-truth reasoning from the dataset (see Section 4.1) are passed to a Curator agent that produces the skill.[2] Skills are stored in a skill directory $\mathcal{S} = \{s_0, \ldots, s_n\}$. We keep skills for different datasets in different directories.

Since no gold set is available, we validate skill quality empirically: for each dataset, we randomly sample 100 queries from the training set and compare two setups, one in which the agent receives only the input query, and one in which it also receives the skill previously generated from that query. On UGPhysics, accuracy improves from 65% to 84%; on PHYSICS, from 63% to 81%. These results confirm that the generated skills encode useful information that agents can leverage to solve complex tasks.

3.3. Skill usage

Skill usage refers to the task of extracting the most relevant and accurate information to answer a new query from a set of skills. Formally, given a test query q and a corpus $\mathcal{C}$ derived from the skill directory $\mathcal{S}$, the goal is to retrieve the most useful information I_q:

$$I_q = f(q, \mathcal{C}) \tag{1}$$

Existing methods (see Section 2) instantiate f as a retrieval procedure over whole skills, taking $\mathcal{C} = \mathcal{S}$:

$$f(q, \mathcal{S}) = \mathrm{Retrieve}(q, \mathcal{S}, k) \tag{2}$$

where $\mathrm{Retrieve}(q, \mathcal{S}, k)$ returns a subset of k skills relevant to query q, via LLM- or embedding-based semantic search.

3.4. Skill-Synthesizer

Our approach instantiates f in two steps, retrieval and synthesis. We build on two main design decisions. First, rather than retrieving whole skills, we segment them into *passages* and retrieve only the most relevant ones. This finer retrieval granularity reduces noise by discarding irrelevant content and focusing the synthesizer's attention on the most pertinent information. Second, instead of concatenating retrieved passages, we synthesize them into a unified, query-specific skill. The goal is to explicitly reason over how retrieved fragments complement each other, and transform them into a coherent, executable skill tailored to the target problem. Formally:

$$f(q, \mathcal{P}) = \mathrm{Synthesize}\left(q, \mathrm{Retrieve}(q, \mathcal{P}, k)\right) \tag{3}$$

[1] The Solver and Judge agent are the same used for testing, and are described in Section 4.2.

[2] More details on the generation procedure in Appendix A.2.

where $\mathcal{C} = \mathcal{P} = \mathrm{Chunk}(\mathcal{S})$ is the set of *passages* obtained by chunking each skill in $\mathcal{S}$ into smaller units (see Section 4.3 for details). Passages are embedded in a vector store to enable targeted retrieval of the skill parts most relevant to the query. As before, k controls the number of retrieved passages.

In the Synthesize phase, the retrieved passages are passed to a Skill-Synthesizer agent that is instructed to group passages by topic, identify concepts across them, and unifies passages inside each group into coherent paragraphs. The agent then reasons over the paragraphs, searching for complementary information. To support this reasoning step, we equipped Skill-Synthesizer with a Python kernel that enables code execution. This tool is especially useful for the scientific tasks we focus on, as it allows the agent to programmatically manipulate numerical information from retrieved passages, verify symbolic formulas, complete dimensional reductions, and check intermediate steps in equation verification.

Finally, the agent creates an augmented skill that synthesizes all the relevant information scattered across the retrieved passages into a concise, query-specific skill. This process not only allows for extracting relevant information while reducing noise, but also generates new knowledge by connecting related pieces of information that were previously represented independently.

4. Experiments

4.1. Benchmarks

We evaluate our approach on UGPhysics (Xu et al., 2025) and PHYSICS (Feng et al., 2025), which are well-suited for studying skill generation and synthesis as they provide challenging, unsaturated physics problems with detailed worked rationales and text-based derivations. Other available benchmarks are less suitable for our setting. CritPt (Zhu et al., 2025) and TPBench (Chung et al., 2025) have less emphasis on reusable worked derivations and do not provide the level of detailed reasoning traces needed for skill generation; PhysReason (Zhang et al., 2025b), PhysicsArena (Dai et al., 2025), and PhysUniBench (Wang et al., 2025) contain substantial multimodal components, whereas our work focuses on textual reasoning trajectories. We provide more details about UGPhysics and PHYSICS below.

UGPhysics includes 5,520 distinct physics problems in 13 domains (e.g., Quantum mechanics). Problems are undergraduate level, usually requiring a specific rule or formula-driven approach to be solved, and involving self-contained derivations. Every datapoint includes the problem statement, the corresponding ground-truth final answer, the answer's physical unit, and the rationale, i.e., a worked solution that explains the necessary steps to solve the problem.

PHYSICS includes 1,297 graduate level physics problems collected from PhD qualifying exams, spanning six domains (e.g., Electromagnetism). Each entry in the dataset includes the problem statement, ground-truth answer, and rationale. Differently from UGPhysics, PHYSICS problems usually require extended derivations, physical intuition, and the integration of concepts across multiple areas of physics. Notably, 66% of the questions are multi-part, with an average of 2.6 subparts per question and a maximum of 14. Each subpart is associated with its own ground-truth final answer. For evaluation, we concatenate the individual subpart answers into a unique answer.

Data Processing As described in Section 3, our methodology requires a train and test split for each dataset in order to generate skills and test them on unseen questions. Since neither UGPhysics nor PHYSICS come with splits, we defined them. The most straightforward option would be randomly assigning datapoints to train/test splits. However, this would not guarantee that the learnings generated during training would be relevant to the test questions — a well known issue in benchmarks evaluating memory systems (Zhou et al., 2026; Mialon et al., 2024; Phan et al., 2025). To control for this, we define a simple splitting approach: given a domain in a dataset (e.g., Electromagnetism in PHYSICS), we cluster problems based on methodological similarity, and split each cluster 80/20 for train and test. In this way, we ensure the training data includes signal that can help with solving questions at test time.[3] Since UGPhysics has a highly skewed distribution of datapoints across domains (min 58, max 1019), we cap the number of datapoints per domain at 300 (approximately the median domain size) and retain random clusters up to this limit, discarding the rest.

We finally obtain 2,674/678 datapoints for train/test split for UGPhysics, and 779/201 for PHYSICS. We run the generation procedure defined in 3.2 on the training splits of each dataset, and generate 1,593 skills for UGPhysics and 571 skills for PHYSICS.

4.2. Pipeline

The experimental pipeline includes three distinct agents: the Retriever agent, the Solver agent, and the Judge agent. Skill retrieval is the focus of this work, and we define the different retriever approaches in Section 4.3. The Solver agent receives a system prompt that instructs it to reason step-by-step on the input query, and includes the retrieved skills.[4] The Solver can run a Python tool to write and run code. The Solver's final output contains its reasoning trace

[3] We used Claude Code (Anthropic, 2026a) for clustering, prompting it to directly read the problems and group them by shared reasoning patterns and solution methods.

[4] The full prompt is in the Appendix A.3.

and final answer. Finally, the Judge agent generates evaluations for the two target metrics (see Section 4.4). For all the agents, we use Anthropic's Sonnet-4-5 (Anthropic, 2025a).

4.3. Approaches

We compare Skill-Synthesizer against several other approaches. Since we only compare skill retrieval, we keep the set of skills and the Solver agent fixed across all methods (see Section 4.2).

Memoryless No skill is provided to the Solver agent. This serves as the baseline against which we measure improvements from learned skills.

LLM Retrieval An LLM is given the name of all skills created at training time and selects up to the top-k most relevant ones (k=1, 3, 5).[5] This mirrors the default skill discovery mechanism in Anthropic's Agent Skills framework (Anthropic, 2026b).

Embedding Retrieval Skills and test queries are embedded and the top-k skills are retrieved by cosine similarity (k=1, 3, 5), using text-embedding-3-small (OpenAI, 2024) with dimension 1536. Embedding-based retrieval is a common approach adopted in skill-based memory systems such as SkillRL (Xia et al., 2026).

Skill-Synthesizer We create passages by chunking original skills into chunks of 512 characters with 64 characters of overlap. Since passages are shorter than full skills, we use a larger k=25 at retrieval time to ensure sufficient coverage.[6]

4.4. Metrics and Evaluation

The Judge agent assigns a binary score to the Solver agent's final answer, as well as *reasoning soundness*, which evaluates the quality of the agent's reasoning. Since the only difference across settings is the retrieval strategy used to access the skill set, reasoning quality serves as an indirect proxy for the quality of the retrieved information. To assess judgment consistency, we run it on a stratified sample of 100 questions three times with responses held fixed, observing a mean per question pairwise agreement of 0.98 for both accuracy and reasoning soundness. One of the authors manually validated the judge's responses on a subset of 20 questions, finding one disagreement.

	Accuracy		Reasoning Sound	
	UGP.	PHY.	UGP.	PHY.
Memoryless	59.5±1.1	52.7±3.4	76.8±1.7	76.8±2.5
LLM (k=1)	62.8±2.1	53.5±2.8	78.5±0.6	76.8±2.4
LLM (k=3)	63.0±1.4	54.4±2.9	77.6±0.3	76.5±1.6
LLM (k=5)	64.2±0.8*	53.7±1.5	77.1±1.4	74.9±1.3
Emb (k=1)	62.3±0.2	51.4±1.7	78.1±0.3	76.1±0.9
Emb (k=3)	64.7±1.7*	52.0±2.0	78.5±0.4	77.2±1.9
Emb (k=5)	65.0±1.0*	54.0±1.6	77.6±0.3	77.3±1.4
Skill-Synt.	**69.5**±0.8*	**63.5**±1.5*	**86.0**±0.7*	**85.9**±0.8*

Table 1. Results on **UGP**hysics and **PHY**sics. ± indicates standard deviation; '*' indicates statistically significant improvement over Memoryless. Best performing setup in bold.

5. Results

Table 1 shows the results on UGPhysics and PHYSICS. We run each setup 3 times and report variance. Statistical significance is determined using McNemar's exact test with Bonferroni correction over conditions within each dataset which has been recommended for comparing classifiers on paired binary outcomes (Raschka, 2020).

Across both datasets, our approach consistently achieves the best performance in both accuracy and reasoning soundness, outperforming the Memoryless baseline and other retrieval approaches. On UGPhysics, Skill-Synthesizer shows a 10-point improvement in accuracy over Memoryless, almost double the improvement of Embedding Retrieval with k=3 and k=5, and LLM Retrieval with k=5. On PHYSICS the difference is even more striking: Skill-Synthesizer again shows a 10-point improvement, while none of the other retrieval approaches lead to a significant improvement. On reasoning soundness, Skill-Synthesizer is the only approach showing a significant improvement over the Memoryless baseline, with a 9 point improvement on both datasets.

The stronger performance of competing retrieval methods on UGPhysics than on PHYSICS likely reflects differences in both skill coverage and task structure. UGPhysics contains roughly three times as many training-time skills as PHYSICS (see Section 3.2) and more narrowly scoped procedural problems, which are more easily retrievable for whole-skill approaches. Conversely, PHYSICS contains more conceptually broad questions requiring derivation, physical insight, and the integration of information across multiple sources (see Section 4.1). Existing retrieval approaches struggle with this combination of data sparsity and multi-source synthesis, while, as we show in Section 6,

[5] We also experimented by giving the LLM the name+description and observed similar performance.

[6] We experimented also with k=10 and 50, observing a decrease in performance for the former and equal performance for the latter.

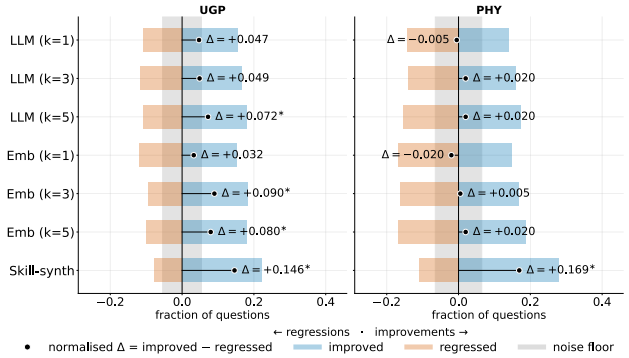

Figure 2. Normalised per-question improvement compared to the memoryless baseline. Blue and orange bars denote the fraction of questions that improved or regressed. The shaded gray band is the expected run-to-run noise floor (see Appendix B.1 for details); Black markers indicate the normalised net improvement. * denotes statistical significance.

Skill-Synthesizer successfully handles them.

6. Analysis

In this section, we investigate the reasons underlying the performance observed in Table 1.

6.1. Impact of Retrieval on Performance

We first analyze the performance differences between the Memoryless baseline and the retrieval-based approaches. Figure 2 shows that, across both datasets, all retrieval approaches lead to improvements on some questions and regressions on others relative to Memoryless.

As expected, gains come from providing task-specific details that the agent lacks. For example, on a UGPhysics problem involving the electric field induced by a charge configuration inside a grounded conducting shell (`classical_electromagnetism_329`), the memoryless agent applies the correct general formula but misses a boundary correction term required by the grounded condition, yielding a wrong coefficient. The retrieval approaches supply the complete expression that accounts for this correction, from which the correct answer follows directly.

Regressions happen because retrieval introduces misleading context that strongly biases the Solver towards an existing solution, a known problem in memory-based approaches (Zhang et al., 2025a; Suzgun et al., 2026; Xia et al., 2026). For example, on a thermodynamics problem in UGPhysics involving gas discharge (`thermodynamics_3431`), the memoryless agent correctly computes the answer. The retrieval approaches, however, surface a related but subtly different scenario, and the agent imports a solution strategy that does not quite apply to the problem at hand, leading it to track the wrong quantity and arrive at an incorrect answer.

6.2. Skill-Synthesizer Performance Analysis

In line with the results in Table 1, Fig. 2 shows that Skill-Synthesizer has the highest delta on UGPhysics and is the only one with a significant positive delta on PHYSICS. This is due to two main reasons.

Fine-grained Retrieval First, thanks to the finer retrieval granularity, Skill-Synthesizer is able to surface the exact parts of the skills containing the relevant information, even when the overall skill document would not rank top-n by full-skill similarity.

For example, a PHYSICS query (`atomic_1_47`) asks for the probability that an electron remains in the ground state after a β-decay event by computing an overlap-integral. Whole-skill retrieval approaches fail here, as they surface skills on things like time-dependent perturbation theory, which are semantically related to "decay", but rely on the wrong physical framework. The correct method lives in a single passage of a skill whose global embedding does not rank highly because β-decay is not prominently mentioned at the document level. Passage-level retrieval directly surfaces this subsection because it contains the exact overlap-integral expression and how to evaluate it. This example shows that, by operating at finer granularity, passage-level retrieval can recover useful procedural knowledge even when it belongs to a skill whose global embedding does not have high similarity with the current query.[7]

Synthesis Skill-Synthesizer is crucial because it does not merely concatenate retrieved passages: it restructures them into a coherent, problem-specific, executable skill. For example, a UGPhysics query (`atomic_1450`) asks for two counting factors, N_W and N_i, that appear in a resonance cross-section formula. The answer requires combining three facts that live in separate retrieved passages into the multiplication $N_W = 3$ and $N_i = 2 \times 2 \times 3 \times 3 = 36$. Importantly, no single passage contains all three facts, nor how they should be combined. Skill-Synthesizer assembles the relevant fragments, explicitly reasons over how they relate to each other, and derives that only a small fraction (3 out of 36) of the possible initial configurations lead to the desired outcome, yielding $N_W/N_i = 1/12$. Crucially, this conclusion requires understanding the role of each factor and how they multiply together. This example shows that, by synthesizing across passages rather than selecting among them, Skill-Synthesizer can reconstruct multi-step arguments whose components are individually incomplete.[8]

We note that Skill-Synthesizer outperforms the other retrieval approaches even in cases where they all successfully recover the relevant skill. For example, in

[7] See the full example in Appendix B.2.

[8] See the full example in Appendix B.3.

	Accuracy		Reasoning Sound	
	UGP.	PHY.	UGP.	PHY.
passage-only	63.4±0.5*	54.9±0.6	77.9±0.4	75.2±2.6
Skill-Synt.	69.5±0.8*	63.5±1.5*	86.0±0.7*	85.9±0.8*

Table 2. The results on **UGP**ysics and **PHY**sics. ± indicates variance, * indicates statistically significant improvements over Memoryless

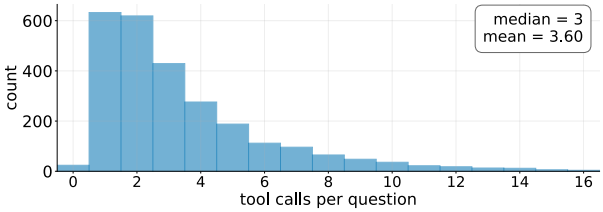

Figure 3. Tool call distribution

a harmonic oscillator stiffening problem in UGPhysics (`quantum_mechanics_4295`), all retrieval methods correctly recover the relevant skill. However, the skill contains the necessary physical ingredients alongside additional heuristics related to a similar yet distinct sub-problem. This extraneous information misleads the other methods, which import the wrong solution strategy and arrive at an incorrect answer. Conversely, Skill-Synthesizer filters out the noise and builds the derivation using only the relevant information, correctly solving the problem.[9]

6.3. Passage Embedding Ablation

To test our intuition about the importance of the synthesis phase, we run a *passage-only* ablation study, in which we remove the Skill-Synthesizer agent and directly pass the concatenated passages to the Solver agent. Results are reported in Table 2.

The passage-only setup improves over Memoryless and some competing baselines on UGPhysics, confirming that providing fine-grained retrieved information is more effective than retrieving entire skills. Crucially, however, passage-only performs substantially worse than Skill-Synthesizer on both datasets, and does not improve over the Memoryless baseline on the more challenging problems in PHYSICS (see Section 4.1). This confirms that providing relevant but raw information is not sufficient to improve performance, and that the gains of Skill-Synthesizer stem from its ability to reason over retrieved content, reconstruct implicit intermediate steps, and adapt partially specified procedures to the constraints of the target problem.

6.4. Impact of Python Tool on Skill-Synthesis

The Python code tool enables Skill-Synthesizer to perform numerical computations and symbolic manipulation to re-derive and verify intermediate steps.

Skill-Synthesizer makes a median of 3 tool calls per question (see Figure 3). An analysis of the calls shows that in the majority of the cases (60%), the tool is used for consistency checks that re-derive or verify intermediate quantities. In 30% of the cases, the tool is used to import `SymPy` and perform symbolic manipulation, including simplification, factorization, and equation solving. General numerical exploration and scratch computation (20%) and direct numerical evaluation used in the final answer computation (15%) are the remaining main usage patterns.[10]

In our analysis, we observe that tool usage is particularly beneficial for derivation-heavy questions. Of the 30 UGPhysics questions where our approach is the only method that produces the correct answer in at least two runs, about two-thirds (19) fall into this category. For these questions, Skill-Synthesizer dedicates most of its tool calls to verifying intermediate quantities and checking the consistency of symbolic manipulations.[11] This information proves crucial for the Solver agent to derive the correct final answer. By contrast, when using the other retrieval methods, the Solver frequently either skips algebraic steps or commits to an incorrect intermediate result without revisiting it, ultimately leading to an incorrect final answer. These results highlight that the Python tool and the model's reasoning capabilities play complementary roles: the model guides the derivation and identifies relevant intermediate steps, while the tool provides reliable symbolic manipulation and consistency verification over the retrieved information.

6.5. Error Analysis

We identified two main kind of errors. First, on simple lookup-style questions where the correct answer just requires reading a table or applying a standard formula, Skill-Synthesizer sometimes over-reasons. For example, for a question about which physical process dominates at a given energy (UGPhysics `atomic_888`), every other baseline simply reads the relevant table from the skill and answers correctly, while Skill-Synthesizer talks itself into selecting the wrong process on spurious grounds. Similarly, in another case (UGPhysics `atomic_1227`), the Skill-Synthesizer wrongly re-derives a standard energy formula, whereas other methods simply apply the retrieved expression. These regressions show that the same synthesis pass that leads to improvements on hard questions (see Section 6.2) can introduce risks on easier ones.

[9] See full example in Appendix B.4.

[10] Note that usage patterns are non-exclusive. See Appendix B.5.1 for examples of each usage type.

[11] A detailed example is provided in Appendix B.5.2.

A second error type concerns ungroundedness. In some cases, Skill-Synthesizer complements retrieved passage content with knowledge drawn from the LLM's parameters, effectively filling gaps in retrieved information. While the boundary between legitimate generalization over retrieved skills and ungrounded extrapolation from the model's own knowledge is subtle, the risk here is hallucination: the synthesized skill does not distinguish parametric knowledge from grounded retrieved content, making errors harder to detect and attribute.

7. Conclusions

We presented Skill-Synthesizer, a novel approach that retrieves relevant passages from a skill library and synthesizes them into a compact, query-specific skill through explicit reasoning and code-assisted verification.

We evaluated Skill-Synthesizer on two challenging physics benchmarks, showing that it consistently outperforms all retrieval baselines in both accuracy and reasoning soundness. The gains are particularly strong on the challenging PHYSICS problems, where no other retrieval method significantly improves over the Memoryless baseline. Our passage-only ablation further isolates the role of synthesis, showing that providing fine-grained passages alone is not sufficient. This suggests that maximizing the usefulness of knowledge in skills requires more than retrieval: it requires the ability to filter irrelevant information and dynamically connect complementary knowledge based on the input query. This property is critical for scientific reasoning, where solving a problem often requires combining theoretical principles, domain-specific heuristics, and procedural knowledge from multiple sources.

Our analysis further highlights the importance of code execution as an integral part of the synthesis process: by verifying intermediate steps and performing symbolic manipulation programmatically, Skill-Synthesizer produces more reliable synthesized skills. This is especially valuable for the derivation-heavy problems that characterize scientific workflows.

Future work will explore several directions, including addressing the over-reasoning and ungroundedness failures discussed in Section 6.5, introducing more rigorous formal verification mechanisms to bound and validate the synthesizer's reasoning, investigating cross-dataset portability of learned skills, and extending the approach to scientific domains beyond physics. More broadly, we expect Skill-Synthesizer to become increasingly valuable as scientific agents accumulate larger and more diverse skill libraries.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., Opsahl-Ong, K., Singhvi, A., Shandilya, H., Ryan, M. J., Jiang, M., et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Anthropic. Claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, 2025a. Accessed: 2026-05-20.

Anthropic. Equipping agents for the real world with Agent Skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025b. Anthropic Engineering Blog.

Anthropic. anthropics/skills: Public repository for Agent Skills. https://github.com/anthropics/skills, 2025c. Accessed: 2026-01-24.

Anthropic. Claude Code documentation. https://code.claude.com/docs/en/gitlab-ci-cd#claude-md-configuration, 2026a. Accessed: 2026-01-20.

Anthropic. Claude Code skills. https://code.claude.com/docs/en/skills, 2026b. Accessed: 2026-05-26.

Chung, D. J., Gao, Z., Kvasiuk, Y., Li, T., Münchmeyer, M., Rudolph, M., Sala, F., and Tadepalli, S. C. Theoretical physics benchmark (tpbench)—a dataset and study of ai reasoning capabilities in theoretical physics. *Machine Learning: Science and Technology*, 6(3):030505, 2025.

Dai, S., Yan, Y., Su, J., Zihao, D., Gao, Y., Hei, Y., Li, J., Zhang, J., Tao, S., Gao, Z., et al. Physicsarena: the first multimodal physics reasoning benchmark exploring variable, process, and solution dimensions. *arXiv preprint ArXiv:2505.15472*, 2025.

Feng, K., Zhao, Y., Liu, Y., Yang, T., Zhao, C., Sous, J., and Cohan, A. Physics: Benchmarking foundation models on university-level physics problem solving, 2025. URL https://arxiv.org/abs/2503.21821.

Hong, K., Troynikov, A., and Huber, J. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma, July 2025. URL https://trychroma.com/research/context-rot.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., et al. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Li, X., Li, M., Bao, K., Ma, Y., Wang, W., Liu, D., and Feng, F. Skillgraph: Skill-augmented reinforcement learning for agents via evolving skill graphs. *arXiv preprint arXiv:2605.12039*, 2026.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Mialon, G., Fourrier, C., Wolf, T., LeCun, Y., and Scialom, T. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. Openai embeddings api. https://platform.openai.com/docs/guides/embeddings, 2024. Accessed: 2026-05-20.

Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., and Khattab, O. 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.

Phan, L., Gatti, A., Han, Z., Li, N., Hu, J., Zhang, H., Zhang, C. B. C., Shaaban, M., Ling, J., Shi, S., et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Raschka, S. Model evaluation, model selection, and algorithm selection in machine learning, 2020. URL https://arxiv.org/abs/1811.12808.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Suzgun, M. and Kalai, A. T. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024.

Suzgun, M., Yuksekgonul, M., Bianchi, F., Jurafsky, D., and Zou, J. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 7080–7106, 2026.

Toscano, J. D., Chai, Z., and Karniadakis, G. E. GRAFT-ATHENA: Self-improving agentic teams for autonomous discovery and evolutionary numerical algorithms. *arXiv preprint arXiv:2605.11117*, 2026.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wang, J. Memento-ii: Learning by stateful reflective memory. *arXiv preprint arXiv:2512.22716*, 2025.

Wang, L., Su, E., Liu, J., Li, P., Xia, P., Xiao, J., Zhang, W., Dai, X., Chen, X., Meng, Y., et al. Physunibench: an undergraduate-level physics reasoning benchmark for multimodal models. *arXiv preprint arXiv:2506.17667*, 2025.

Xia, P., Chen, J., Wang, H., Liu, J., Zeng, K., Wang, Y., Han, S., Zhou, Y., Zhao, X., Chen, H., et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Xu, X., Xu, Q., Xiao, T., Chen, T., Yan, Y., Zhang, J., Diao, S., Yang, C., and Wang, Y. UGPhysics: A comprehensive benchmark for undergraduate physics reasoning with large language models. *arXiv preprint arXiv:2502.00334*, 2025.

Ye, H., He, X., Arak, V., Dong, H., and Song, G. Meta context engineering via agentic skill evolution. *arXiv preprint arXiv:2601.21557*, 2026.

Zhang, Q., Hu, C., Upasani, S., Ma, B., Hong, F., Kamanuru, V., Rainton, J., Wu, C., Ji, M., Li, H., et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025a.

Zhang, X., Dong, Y., Wu, Y., Huang, J., Jia, C., Fernando, B., Shou, M. Z., Zhang, L., and Liu, J. Physreason: A

comprehensive benchmark towards physics-based reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16593–16615, 2025b.

Zhou, H., Guo, S., Liu, A., Yu, Z., Gong, Z., Zhao, B., Chen, Z., Zhang, M., Chen, Y., Li, J., et al. Memento-Skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

Zhu, M., Tian, M., Yang, X., Zhou, T., Yuan, L., Zhu, P., Chertkov, E., Liu, S., Du, Y., Ji, Z., et al. Probing the critical point (critpt) of ai reasoning: a frontier physics research benchmark. *arXiv preprint arXiv:2509.26574*, 2025.

A. Skill

A.1. Example Skill

Below we provide an example of a generated skill. The skill is stored as a structured markdown document and is retrieved at inference time when the system identifies semantic or procedural similarity between the current query and previously observed trajectories.

Example Skill: maxwell-equations-with-magnetic-monopoles.md

Description: Strategy for writing Maxwell's equations when magnetic monopoles are present. Use when a problem involves magnetic charge density ρ_m, magnetic current density $\mathbf{j}_m$, electromagnetic duality, or generalized Maxwell equations with both electric and magnetic sources.

Core Equations:

$$\nabla \cdot \mathbf{D} = \rho$$

$$\nabla \cdot \mathbf{B} = \rho_m$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} - \mathbf{j}_m$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{j}$$

Key Reasoning Patterns:

- Use the $\mathbf{D}$ and $\mathbf{H}$ formulation rather than the $\mathbf{E}$ and $\mathbf{B}$ formulation.
- The magnetic current term $-\mathbf{j}_m$ appears in Faraday's law.
- Magnetic charge density appears in
$$\nabla \cdot \mathbf{B} = \rho_m.$$

Common Mistakes:

- Forgetting the magnetic current term in Faraday's law
- Placing ρ_m in the wrong divergence equation
- Using the vacuum-only formulation when a general form is requested

This example illustrates several properties of our skill representation. First, the skill stores procedural and conceptual information rather than memorizing a single question-answer pair. Second, the skill explicitly captures both successful reasoning patterns and common failure modes observed in previous trajectories. Finally, the skill is written in a human-readable format, allowing both direct model conditioning and external inspection or editing. In practice, the repository contains many such skills spanning different scientific domains and reasoning strategies, which are retrieved and synthesized dynamically during inference.

A.2. Skill Generation

Here we detail the skill-generation process. Initially, the Solver is given a query and produces a solution attempt, which is evaluated by the Judge. The Solver's response and evaluation are then passed to the Curator, which additionally has access to the rationale underlying the gold solution (see Section 4.1), i.e., the sequence of reasoning steps required to derive the correct answer. The Curator analyzes the trajectory and can either create or update one or more skills containing the most relevant lessons extracted from the attempt, which is then stored in the centralized skill repository. Importantly, the Curator can also do nothing if a relevant skill already exists.

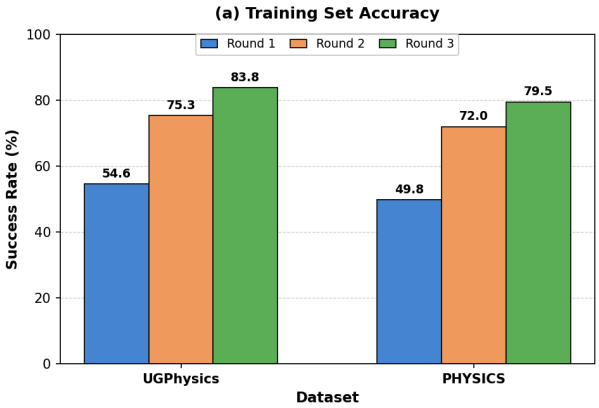

Figure 4. Per-round training accuracy on UGPhysics and PHYSICS.

We adopt an online learning setting in which the Solver solving query k has access to skills generated from queries 0 through $k-1$. Furthermore, the Curator is allowed not only to create new skills, but also to refine or update previously generated skills as additional trajectories are observed. This setting more closely mirrors real-world deployment, where agents continuously accumulate and refine knowledge over time rather than learning from a fixed offline corpus.

Finally, for each query, we allow the Solver up to three attempts. After each failed attempt, newly generated skills become immediately available to subsequent attempts on the same query, enabling the Solver to iteratively reason over and learn from its own prior failures. This iterative refinement process substantially increases the likelihood that the Solver converges to a correct solution while also producing richer trajectories for skill generation. We report the per attempt accuracy on the training set in figure 4.

A.3. Solver Prompts

The Solver agent receives a system prompt that instructs it to solve the problem while leveraging any retrieved skills. The prompt additionally specifies the expected output format, code-execution behavior, and usefulness annotation used throughout our experiments.

System Prompt Used by the Solver Agent

```
You are a problem solver. Your goal is to solve the given
question accurately and efficiently, leveraging any relevant
skill reference materials provided.

## Instructions

1. Read the Question First
   - Carefully read and understand the question below
     before looking at any skill materials.
   - Identify the problem type, key variables,
     and what is being asked.

2. Review Skills & Plan
   - After understanding the question, review the
     RELEVANT SKILLS section.
   - Identify which skills or patterns are most
     relevant to the current question.
   - Plan your approach before diving into the solution.

3. Solution Development
   - Show clear, logical steps in your reasoning.
   - If a skill provides a relevant strategy,
```

```
     see if you can apply it.
   - Verify intermediate results where possible.

4. Programming Tasks
   - When computation is needed, write clean Python code
     in a ```python block.
   - Always include print statements to display results.
   - After writing code, type EXECUTE CODE!
     on a new line to trigger execution.
   - Use the code output to verify and refine your answer.

5. Final Answer
   - Present your final answer clearly using:

   FINAL ANSWER:
   <answer>YOUR ANSWER HERE</answer>

6. Skill Usefulness
   - After your final answer, rate how useful
     the retrieved skill was:

   <skill_usefulness>
   high/medium/low/none
   </skill_usefulness>
```

The user message contains the target question. When skills are placed in the user context rather than the system prompt, the retrieved skills are appended below the question in a dedicated `RELEVANT SKILLS` section.

User Prompt Template

```
## QUESTION

[[QUESTION]]

---

## RELEVANT SKILLS

"""
[[SKILLS]]
"""
```

B. Analysis

B.1. Statistical Significance Computation

Each condition is run three times on the same set of N questions, producing three binary correctness vectors of length N. For each of the three unordered run-pairs (i, j) within a condition, the disagreement rate is the fraction of questions on which runs i and j produce different outcomes; the condition's noise estimate is the median of those three rates. The dataset's pooled noise is the median of the per-condition estimates, giving 12.9% on PHYSICS and 10.9% on UGPhysics. Because two same-condition runs are exchangeable, a question on which they disagree is equally likely to look like an improvement or a regression when one is held out as the "baseline," so we halve the pooled rate to obtain the directional noise floor used in Fig. 2 (6.5% on PHYSICS, 5.4% on UGPhysics); Δ values whose improvement and regression components both lie inside the band are consistent with run-to-run noise.

B.2. Passage- vs. Skill-Retrieval

Example: Passage-Level Retrieval vs. Whole-Skill Retrieval (`atomic_1_47`)

Problem. The question asks for the probability that an electron remains in the ground state after a Tritium β-decay event instantaneously changes the nuclear charge from

$$Z = 1 \quad \rightarrow \quad Z = 2.$$

Solving the problem requires applying the sudden approximation and evaluating the overlap probability

$$P = \left| \left\langle \psi_f^{(Z=2)} \middle| \psi_i^{(Z=1)} \right\rangle \right|^2 .$$

Failure of Whole-Skill Retrieval. Embedding-based whole-skill retrieval consistently ranks irrelevant skills in the top retrieval positions due to superficial semantic overlap with terms such as "hydrogen atom" and "decay." The retrieved skills primarily concern:

- time-dependent perturbation theory,
- electric dipole transitions,
- generic atomic decay processes.

Although semantically related, these skills rely on the wrong physical framework and do not contain the overlap-integral formalism needed for the problem.

Why Passage Retrieval Succeeds. The correct solution method is contained inside a specific passage of the skill

`sudden-potential-change-orbital-transitions`,

whose overall document description does not explicitly mention β decay and therefore does not rank highly under whole-skill similarity retrieval.
However, passage-level retrieval directly surfaces the relevant subsection because it contains the exact overlap-integral expression

$$P = \left| \left\langle \psi_f^{(Z=2)} \middle| \psi_i^{(Z=1)} \right\rangle \right|^2$$

together with the hydrogenic wavefunctions required to evaluate it.

Key Observation. This example illustrates a central limitation of whole-skill retrieval: useful procedural information may exist only within a small localized portion of a larger skill document whose global semantic embedding is dominated by unrelated concepts. Passage-level retrieval avoids this failure mode by operating at a finer granularity, allowing the system to recover highly relevant derivation fragments even when the surrounding skill document would not rank among the top retrieved results.

B.3. Skill-Synthesizer: Cross-Passage Synthesis

Example: Degeneracy Factors in W^+ Production (`atomic_1450`)

Problem. In the quark model, W^+ bosons are produced at a $\mathrm{p\bar{p}}$ collider via the subprocess $u + \bar{d} \rightarrow W^+ \rightarrow e^+ + \nu$. The Breit-Wigner cross-section for this process contains the ratio

$$\frac{N_W}{N_i},$$

where N_W counts the internal configurations of the produced W^+ and N_i counts those of the initial $u + \bar{d}$ pair. The problem asks for the magnitudes of N_W and N_i.

Ground Truth.

$$N_W = 3, \qquad N_i = 36, \qquad \frac{N_W}{N_i} = \frac{1}{12}.$$

$N_W = 3$ because the W^+ is a massive spin-1 particle with three polarization states and no color charge. $N_i = 36$ because each of the two initial-state quarks carries two independent spin orientations *and* three independent color charges:

$$N_i = \underbrace{2 \times 2}_{\text{spin}} \times \underbrace{3 \times 3}_{\text{color}} = 36.$$

Observed Behavior Across Methods. All approaches correctly recover $N_W = 3$. All non-Skill-Synthesizer methods fail on N_i, producing one of three distinct wrong answers:

Approach	N_i	Error
Full-skill (top-3, top-5)	3	counts only color-singlet pairs
Name-only, passage-embedding	4	counts spin only (2×2), ignores color
Memoryless (2/3 runs)	9	counts color only (3×3), ignores spin

In contrast, all Skill-Synthesizer variants produce the correct $N_i = 36$ across all 9 runs (3 variants $\times$ 3 runs).

Retrieved Information. All approaches retrieve passages from similar skills covering gauge-boson properties, weak-interaction vertices, and collider cross-sections. The facts needed to compute the answer correctly are split across three distinct passages:

- one passage establishes that the W^+ is a massive spin-1 boson with **3 polarization states** and no color charge, fixing $N_W = 3$ (from a skill on gauge-boson structure);
- a second passage confirms that $u + \bar{d} \to W^+$ is a valid production vertex, **identifying the two initial-state particles** as spin-$\frac{1}{2}$ quarks (from a skill on weak-interaction charged-current vertices);
- a third passage establishes that each quark and antiquark carries **3 independent color charges** (from a skill on quark counting in collider cross-sections).

No single retrieved passage states that N_i requires combining all spin and color degrees of freedom of both initial-state particles into the product $2 \times 2 \times 3 \times 3$.

Why Synthesis Matters. Non-Skill-Synthesizer methods read passages independently and anchor on whichever factor is most prominently featured in their highest-ranked passage. Approaches that retrieve a passage emphasizing color produce $N_i = 3$ or 9; those retrieving a passage emphasizing spin produce $N_i = 4$. Neither recognizes that the formula requires averaging over *all* initial-state degrees of freedom simultaneously.
Skill-Synthesizer explicitly reasons over the role of each retrieved fact: it identifies N_i as a product over *all* internal degrees of freedom of *all* initial-state particles, constructs the full multiplication $2 \times 2 \times 3 \times 3 = 36$, and interprets the ratio $N_W/N_i = 1/12$ as the fraction of initial configurations that can resonantly produce the W^+.

Key Observation. This example illustrates that the bottleneck is not retrieval quality but *compositional reasoning*: the answer requires combining three independently retrieved facts in the right way. Retrieval-only approaches implicitly treat each passage as a self-contained answer template and apply whichever one ranks highest, missing the cross-passage multiplication. Skill-Synthesizer, by constructing a unified derivation before generating the final answer, is forced to combine all three facts and arrives at the correct result.

B.4. Skill-Synthesizer: Constraint-Based Reasoning

Example: Parity-Constrained Revival Dynamics (`quantum_mechanics_4295`)

Problem. A particle with mass m is initially in the ground state of the harmonic oscillator potential

$$V_1(x) = \frac{1}{2}Kx^2.$$

The potential is suddenly changed to

$$V_2(x) = Kx^2,$$

and no measurement is made. After a time interval τ, the potential is switched back to V_1. The problem asks for the values of τ such that the particle returns to the ground state of V_1 with probability 100%.

Ground Truth. The correct family of solutions is

$$\tau = l\pi\sqrt{\frac{m}{2K}}, \qquad l = 1, 2, \ldots$$

corresponding to the half-period

$$\frac{\pi}{\omega_2}$$

of the modified oscillator.

Observed Behavior Across Methods. All retrieval-only approaches fail on this problem. Across all 11 non-Skill-Synthesizer configurations and 3 runs per configuration (33 runs total), every method predicts

$$\tau = \pi\sqrt{\frac{2m}{K}} = \frac{2\pi}{\omega_2},$$

i.e., the full oscillation period of V_2. These methods additionally fail to recover the full family of valid solutions. In contrast, all Skill-Synthesizer variants solve the problem correctly across all runs (9/9 total runs), producing

$$\tau = k\pi\sqrt{\frac{m}{2K}}, \qquad k = 1, 2, \ldots$$

Retrieved Information. All approaches successfully retrieve the relevant skill

`sudden-approximation-harmonic-oscillator-parameter-change`,

which discusses sudden quenches and harmonic oscillator evolution. However, alongside the necessary physical ingredients, the skill also contains a procedural template for a related but distinct sub-problem — computing one-way transition probabilities after a single parameter change — which does not apply here. The retrieval-only methods latch onto this template and apply it directly, predicting the full oscillation period of V_2 as the revival time and missing the correct family of solutions.

Why Synthesis Matters. Rather than applying the retrieved template directly, Skill-Synthesizer reconstructs the full physical argument from the retrieved passages. Specifically, it reasons through:

1. the sudden-overlap expansion into the eigenbasis of V_2,
2. the time evolution of the expansion coefficients,
3. the overlap back onto the original state $|0^{(1)}\rangle$,
4. the parity structure of the evolved state.

By explicitly reasoning over the parity constraint — that the ground state of V_1 has even parity and therefore overlaps only with even-n eigenstates of V_2 — Skill-Synthesizer identifies that the relevant phases realign after the half-period π/ω_2, not the full period $2\pi/\omega_2$, and correctly derives the complete family of solutions.

Key Observation. This example illustrates that retrieval alone is not sufficient even when the correct skill is successfully surfaced. When a skill contains both relevant information and extraneous procedural templates, retrieval-only methods are misled by the noise, while Skill-Synthesizer filters it out and builds the derivation using only what is relevant to the target problem.

B.5. Tool Analysis

B.5.1. Tool Surface and Call Categorization

The Skill-Synthesizer is equipped with a single external tool, `run_python`, which executes Python code and returns the resulting standard output to the model. Skill-Synthesizer frequently uses the Python environment as an intermediate reasoning workspace for symbolic manipulation, derivation verification, and consistency checking across retrieved passages.

To better understand how the Skill-Synthesizer uses this tool during reasoning, we categorize each emitted `(code, output)` pair into one or more semantic categories using lightweight heuristic rules.

Tool Call Categories

- **Consistency Check.** The code or output explicitly verifies agreement between two expressions or quantities. Typical indicators include:

```
isclose, allclose, assert,
verify, check, matches
```

  or limiting-case comparisons and equivalence tests between independently derived expressions.

- **Symbolic Manipulation.** The tool call performs algebraic or symbolic operations using `sympy`. Common operations include:

```
solve, simplify, factor,
expand, integrate, diff,
Matrix, Eq, trigsimp, collect
```

- **Numerical Evaluation.** The tool call performs direct numerical computation using libraries such as:

```
numpy, math, scipy, mpmath
```

  typically producing floating-point outputs, evaluated constants, or final numerical predictions.

- **Exploration.** General-purpose intermediate reasoning that does not fit the previous categories, including scratch calculations, variable substitutions, intermediate plotting, dimensional exploration, or partial derivation work.

The categories are intentionally non-exclusive. A single tool call may simultaneously manipulate symbolic expressions, evaluate numerical quantities, and verify consistency between independently derived formulas. For example, many calls first derive a closed-form symbolic expression using `sympy` and then numerically compare the result against a retrieved approximation or limiting case. We therefore report multi-label statistics rather than collapsing each call into a single mutually exclusive category.

Example: Numerical Evaluation

Question (ugp_cm_1566): compute the escape velocity from the surface of the Moon, given G, M_{moon}, and R_{moon}. The tool call substitutes constants into $v_{\text{esc}} = \sqrt{2GM/R}$ and returns the value in m/s and km/s; the output is then quoted verbatim in the final answer.

```
import numpy as np

G       = 6.674e-11       # m^3 kg^-1 s^-2
M_moon  = 7.35e22         # kg
R_moon  = 1.74e6          # m  (= 1.74e3 km)

v_esc = np.sqrt(2 * G * M_moon / R_moon)
print(f"Escape velocity = {v_esc:.4f} m/s")
print(f"Escape velocity = {v_esc/1000:.4f} km/s")
```

Output:

```
Escape velocity = 2374.5272 m/s
Escape velocity = 2.3745 km/s
```

Example: Exploration

Question (ugp_qm_4287): variational ground state of a Hamiltonian whose minimization yields $E_{\min} = -\frac{49}{16}\frac{me^4}{\hbar^2}$ (obtained in an earlier symbolic step). This call performs a scratch *variable substitution* that re-expresses $E_{\min}$ in units of the hydrogen ground-state energy E_H, providing a sanity-check magnitude rather than producing the final boxed answer.
Note: This call's printed numbers are not used in the boxed final answer. They are an intermediate sanity check that re-expresses an already-derived expression

```
# E_H = -m e^4 / (2 hbar^2)  =>  m e^4 / hbar^2 = -2 E_H
# E_min = -49 m e^4 / (16 hbar^2)
#       = (49/8) * (-m e^4 / (2 hbar^2))
#       = (49/8) * E_H

print("E_min = (49/8) * E_H   where E_H = -13.6 eV")

E_H_eV   = -13.6
E_min_eV = (49/8) * E_H_eV
print(f"E_min = (49/8) * ({E_H_eV} eV) = {E_min_eV:.2f} eV")
print(f"Ratio E_min / E_H = 49/8 = {49/8}")
```

Output:

```
E_min = (49/8) * E_H   where E_H = -13.6 eV
E_min = (49/8) * (-13.6 eV) = -83.30 eV
Ratio E_min / E_H = 49/8 = 6.125
```

An example of consistency check and symbol manipulation can be found in the worked example in B.5.2.

B.5.2. Worked Example: Consistency Check on ugp_atomic_1517

Worked Example: Consistency Check on ugp_atomic_1517

The question asks for

$$\sigma_{\text{tot}}(\text{p}\bar{\text{p}} \to W^+ \to e\nu)$$

at

$$\sqrt{s} = 540 \text{ GeV},$$

given the quark PDFs

$$u(x_p) = 6(1 - x_p)^2, \qquad \bar{d}(x_{\bar{p}}) = 3(1 - x_{\bar{p}})^2.$$

After deriving the parton-luminosity reduction

$$\mathcal{L}(\tau) = 18 \int_\tau^1 \frac{(1-x)^2(x-\tau)^2}{x^3}\, dx, \qquad \tau \equiv \frac{M_W^2}{s},$$

the Skill-Synthesizer postulates the closed form

$$\mathcal{L}(\tau) = -18(\tau^2 + 4\tau + 1)\ln\tau + 54(\tau^2 - 1),$$

and issues a single tool call that both performs the integral symbolically and verifies the proposed expression.

Symbolic Verification Tool Call

```
import sympy as sm
tau, x = sm.symbols('tau x', positive=True)

integrand = 18 * (1-x)**2 * (x - tau)**2 / x**3

result = sm.expand(
    sm.integrate(integrand, (x, tau, 1))
)

postulate = (
    -18*(tau**2 + 4*tau + 1)*sm.log(tau)
    + 54*(tau**2 - 1)
)

print("L(tau) =", result)
print("difference:", sm.simplify(result - postulate))
```

Tool Output

```
L(tau) = -18*tau**2*log(tau)
          + 54*tau**2
          - 72*tau*log(tau)
          - 18*log(tau)
          - 54

difference: 0
```

This call is simultaneously *symbolic* (via `sm.integrate` and `sm.simplify`) and a *consistency check*, since the output `difference: 0` certifies algebraic equivalence between the derived integral and the proposed closed-form expression.
The verified expression is then used in the final numerical evaluation,

$$\sigma \approx 3.5 \times 10^{-34}\,\text{cm}^2, \qquad N \approx 1100,$$

which closely matches the target values

$$3.45 \times 10^{-34}\,\text{cm}^2 \quad \text{and} \quad 1100 \text{ events}.$$

By contrast, the retrieval-only baseline, which lacks symbolic tool access, commits to an incorrect intermediate expression for the parton luminosity and predicts

$$\sigma \approx 5.6 \text{ pb}, \qquad N \approx 18,$$

roughly two orders of magnitude below the target value without revisiting the derivation.