FORGETTING IS WHAT TO LEARN: VERIFIER-GROUNDED MINIMAL-CORE SKILLS FOR SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI for Science (AI4Science) increasingly relies on AI agents to execute long-horizon scientific workflows that require multiple reasoning steps, specialized tools, data transformations, and domain-specific procedural constraints. Reusable skills can transfer useful scientific experience across tasks, but directly accumulating successful workflows also preserves redundant, source-specific, and behaviorally unnecessary procedures. We present AUTOSKILLS, a verifier-grounded framework for evolving reusable skills for long-horizon task execution. Given a successful reference workflow, we construct an over-complete skill and performs verifier-grounded forgetting: candidate operations are removed and the solver is rerun under the same executable verifier, retaining only deletions that preserve task performance. The remained core skill is then generalized for transfer and selectively reused based on empirical evidence. We evaluate primarily on biomedical scientific workflows, where tasks involve multi-step data analysis, domain-specific procedures, and executable evaluation. On a 39-task BioDSBench panel, we achieve 100.0% accuracy, compared with 76.9% without skills and 79.5% with SkillOpt. On BioMNIBench, we improve the mean reward from 0.790 to 0.917, and on a larger 159-task BioDSBench evaluation reaches 94.97% accuracy versus 77.36% without skills. Beyond the primary biomedical setting, skill transfer improves performance on LiveMathematician from 0.57 to 0.68 and on OfficeQA from 0.33 to 0.60.

1 INTRODUCTION

Recent advances in AI for Science (AI4Science) have enabled AI agents to assist with increasingly complex scientific workflows, ranging from biomedical data analysis and statistical modeling to hypothesis testing and computational discovery. Unlike conventional question answering, these scientific tasks often require agents to operate over long horizons, coordinating multiple reasoning steps, invoking specialized tools, transforming intermediate data products, and satisfying domain-specific methodological constraints before producing a valid result. In such settings, scientific success depends not only on whether an agent can generate a plausible answer, but also on whether it can execute a correct and reproducible sequence of procedures. This has motivated recent agentic systems to explore memory, workflow reuse, and self-improvement as mechanisms for carrying useful experience across scientific tasks (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024b; Zhang et al., 2026b;a).

Reusable skills provide a natural interface for transferring such procedural knowledge across long-horizon tasks. Instead of repeatedly rediscovering how to perform a complex workflow, an agent can preserve useful procedures, decision criteria, tool-use patterns and apply them to related tasks. However, this creates a fundamental challenge that is particularly important in realistic long-horizon settings: **Not every step in a successful workflow is necessary for completing the task.** Long executions often contain redundant operations, source-specific assumptions, alternative procedures, and steps that happened to be useful for one instance but are irrelevant to another. If all seemingly successful behavior is permanently stored, skill libraries may grow rapidly while also introducing irrelevant or incompatible behavior into agents' context.

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.

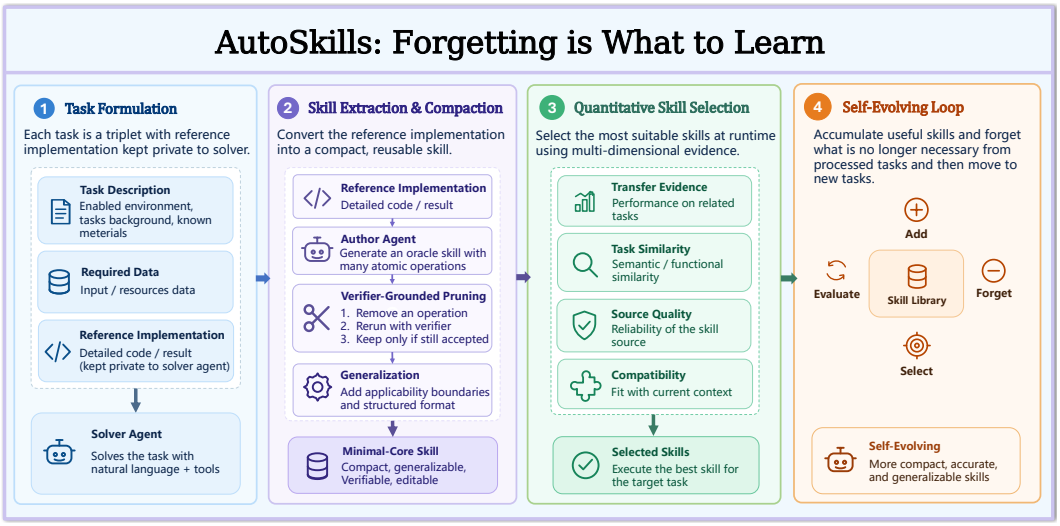

Figure 1: Overview of AUTOSKILLS for long-horizon task execution. Real-world tasks often require agents to coordinate multiple reasoning steps, tools, data transformations, and verification constraints. Given a task with a private reference implementation, an author agent first extracts an over-complete reusable skill. AUTOSKILLS then performs verifier-grounded forgetting by removing candidate operations, rerunning the solver, and retaining only deletions that preserve verified behavior. The resulting core skill is generalized with explicit applicability boundaries and stored in the skill library. At runtime, skills are selected using transfer evidence, task similarity, compatibility, and source quality. This lifecycle enables agents to accumulate reusable procedural knowledge while avoiding the uncontrolled accumulation of unnecessary behavior.

This problem is more than a memory-compression issue, especially for scientific tasks. Scientific workflows are often composed of interdependent analysis steps, where an unnecessary piece of procedural knowledge can affect downstream decisions: it may cause an agent to invoke an inappropriate tool, apply an irrelevant statistical procedure, mis-handle data transformations, or follow source-specific assumptions that do not hold for the target study. At the same time, carrying such content into the agent's context consumes valuable context and execution budget, increasing the risk of distraction or conflicting instructions in long-horizon workflows. Consequently, the central question is not simply how to accumulate more successful trajectories and distill them into skills, but how to identify which parts of successful scientific behavior are genuinely reusable and necessary for future tasks.

We therefore distinguish *successful behavior* from *necessary behavior*. Consider a fixed solver operating on a fixed long-horizon task under an executable verifier. The necessity of an operation can be evaluated by removing that operation, rerunning the same solver, and comparing the resulting performance with that of the original skill. Because the solver, task, evaluation protocol, and verifier remain fixed, the effect of removing the operation can be measured consistently. Under this view, skill acquisition proposes candidate procedural knowledge, whereas skill evolution requires empirical evidence to determine which parts of that knowledge should be retained.

We present AUTOSKILLS, a verifier-grounded framework for skill evolution in long-horizon task execution. Figure 1 summarizes the framework. Starting from a reference implementation of a successful workflow, an author agent constructs an over-complete skill composed of atomic operations. AUTOSKILLS then performs verifier-grounded forgetting: candidate operations are removed and the solver is rerun under the same executable verifier, while failed deletions are rolled back. The accepted artifact is a verifier-screened core rather than a claim of global minimality. The goal is therefore not to find the shortest possible workflow, but to remove procedural content whose behavioral necessity is not supported by execution evidence.

The resulting core is only the first stage of a broader skill lifecycle. **Pruning removes unnecessary behavior**, whereas **generalization removes unnecessary specificity**. After pruning, the surviving operations are rewritten to expose transferable principles, remove source-specific anchors, and

define explicit applicability boundaries. The generalized skill is then validated on held-out tasks. Finally, because not every useful skill should be injected into every long-horizon task, a separate runtime selection mechanism uses transfer evidence, task similarity, source quality, and compatibility constraints to decide when a skill should be reused. This separation allows the framework to address three distinct questions in long-horizon execution: what useful behavior should be retained, what behavior should be forgotten, and when retained behavior should be reused.

We study this lifecycle primarily in biomedical scientific workflows. Biomedical data-science tasks provide a particularly suitable setting for studying long-horizon agent behavior: they require heterogeneous data processing, domain-specific statistical decisions, multiple dependent analysis steps, and tool-mediated execution, while offering executable or otherwise objective evaluation signals. This combination exposes failure modes that are difficult to capture with answer-only benchmarks and makes it possible to test not only whether an agent reaches a correct final result, but whether reusable procedural knowledge improves the execution of a realistic workflow. We therefore use BioMNIBench and BioDSBench (Wang et al., 2024a; Qu et al., 2026) as the primary evaluation environments, while LiveMathematician and OfficeQA(He et al., 2026; Opsahl-Ong et al., 2026) serve as extended fields evaluation.

Our evaluation shows consistent benefits on the primary biomedical setting. On the shared 39 long-horizon tasks BioDSBench panel, the no-skill condition, the SkillOpt(Yang et al., 2026) artifact, and AUTOSKILLS obtain 76.9%, 79.5%, and 100.0% accuracy respectively. On the aligned BioMNIBench evaluation, the no-skill condition obtains a mean reward of 0.790, while AUTOSKILLS obtains 0.917. On a larger 159-task BioDSBench evaluation, AUTOSKILLS reaches 94.97% compared with 77.36% for the no-skill baseline. Extended experiments demonstrate transfer beyond AI for Science: performance improves from 0.57 to 0.68 on LiveMathematician and from 0.33 to 0.60 on OfficeQA. These results suggest that the learned procedural skills capture reusable task-solving knowledge rather than merely compressing task-specific workflow text.

Our contributions are:

- We study **skill evolution for long-horizon scientific workflows** and introduce verifier-grounded forgetting, which removes operations only when their deletion preserves task performance under the same solver and executable verifier.
- We develop a two stage process combining verifier-grounded pruning with transfer-oriented generalization to remove unnecessary and source-specific procedures while retaining reusable scientific knowledge.
- We introduce evidence-based runtime skill selection and demonstrate the framework primarily on biomedical scientific workflows, with additional cross-domain experiments showing transfer beyond AI for Science.

2 RELATED WORK

Memory and self-evolving agents. Reflexion and Self-Refine use language feedback to improve subsequent attempts without changing model weights (Shinn et al., 2023; Madaan et al., 2023). ExpeL and Agent Workflow Memory store reusable experiences or workflows (Zhao et al., 2024; Wang et al., 2024b), while Agentic Context Engineering and the Darwin Gödel Machine extend editable agent state to prompts, code, or agent programs (Zhang et al., 2026b;a). These systems establish the importance of maintaining reusable agent state. In contrast, our focus is the evidence required to determine whether an individual operation in a successful skill is behaviorally necessary.

Automatic skill discovery and optimization. Recent work explores different ways to discover, organize, and improve reusable skills. Trace2Skill distills trajectory-level lessons through parallel consolidation and studies transfer across tasks and models (Ni et al., 2026). EvoSkill analyzes failures, edits structured skills, and uses held-out validation to evaluate candidate artifacts (Alzubi et al., 2026). SkillRL evolves a hierarchical SkillBank through experience distillation, retrieval, and reinforcement learning (Xia et al., 2026). SkillOpt treats a skill as a text document and optimizes it through bounded add, delete, and replace edits accepted by held-out validation (Yang et al., 2026). Our framework addresses a different level of the problem. Rather than only optimizing a complete skill artifact, it constructs candidate content from a reference implementation and tests the necessity

of individual atomic operations through behavioral intervention. The pruned skill is then generalized for transfer and selected using evidence about the target task. Thus, our method is complementary to skill discovery and optimization systems, while emphasizing the distinction between artifact-level usefulness and operation-level necessity.

Verification and benchmarked agent execution. Executable tests and environment feedback provide stronger behavioral evidence than model self-assessment when tasks admit objective or executable evaluation procedures (Chen et al., 2022; 2024; Lightman et al., 2024). LLM-based judges, in contrast, can introduce systematic biases and inconsistencies (Zheng et al., 2023). Recent research-repository benchmarks evaluate end-to-end agent execution on complex tasks (Bogin et al., 2024; Starace et al., 2025; Siegel et al., 2024). However, end-to-end benchmarks typically evaluate whether an agent can produce a successful final outcome, without assuming that a reusable, directly learnable workflow is available from the task itself. Our work instead focuses on *skill learning and self-evolution* in realistic scientific workflows, where useful procedures must be extracted, compressed, generalized, and subsequently reused across related tasks. To evaluate this setting beyond conventional end-to-end datasets, we place particular emphasis on biomedical data-science workflows, where tasks involve executable analyses and domain-specific tool use, using BioDSBench and BioMNIBench as complementary evaluation environments.

3 METHOD

AUTOSKILLS consists of four stages: reference-grounded authoring constructs an over-complete oracle skill (§3.2); verifier-grounded pruning tests the behavioral necessity of each operation (§3.3); generalization removes source-specific details (§3.4); and runtime selection decides whether a skill is injected into a target task (§3.5). Full harness details, the complete pruning procedure, the generalization checks, and the selector specification are deferred to Appendix A.

3.1 PROBLEM SETTING AND MULTI-ROUND EVALUATION

We represent each task as $t = (\pi_t, D_t, C_t, V_t, H_t)$, where π_t is a problem description derived from a research artifact, D_t is the data visible to the solver, C_t is a reference implementation available only to the skill author, V_t is the private executable judge, and H_t is the public-output validator. The solver receives (π_t, D_t) and an optional rendered skill, but cannot access C_t, the private judge, or the reference outputs. The reference implementation is used to construct reusable knowledge; it is never copied into the solver's workspace.

Let M denote the frozen language model and let S denote the skill injected into the solver context. A skill is an ordered collection of atomic operations $S = \{o_1, \ldots, o_n\}$, where each operation contains a solver-facing instruction, an operation identifier, dependency metadata, a role, and any owned resource or script. The role distinguishes reasoning operations from plumbing operations such as file preparation and submission conventions.

The harness evaluates a task through multiple one or more rounds: at each round, the solver receives the public task, the rendered skill, and self-reflection(Shinn et al., 2023) from previous rounds (if failed to get full marks); its candidate output is first checked by the public validator H_t and then scored by the private judge, $(y_{t,r}, f_{t,r}) = V_t(z_{t,r})$ with $y_{t,r} \in [0, 1]$ (Appendix A.1). Let $r^\star$ denote the final valid judge attempt in a run. The task-level utility of a skill is

$$y_t(S) = y_{t,r^\star}, \qquad U_{\mathcal{T}}(S) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} y_t(S), \tag{1}$$

where $\mathcal{T}$ is a fixed task split evaluated under one benchmark protocol. All comparisons between skills use the same model, temperature, reasoning mode, round budget, timeout, task set, validator, and judge. The no-skill condition is denoted by $S = \varnothing$ and provides a capability baseline. It is not used as a substitute for the private verifier gate.

The quantity of interest is the necessity of an operation relative to a solver, a task split, and a verifier. For an operation o_i and a current skill S, define

$$\Delta_i(S, \mathcal{T}) = U_{\mathcal{T}}(S) - U_{\mathcal{T}}(S \setminus \{o_i\}). \tag{2}$$

A positive value indicates that removing o_i lowers utility, whereas a value close to zero indicates that the operation is not required under the tested protocol. This quantity is behavioral: it is not an intrinsic property of the operation text.

3.2 Reference-Grounded Skill Authoring

The author agent G receives the public task description, visible data, and private reference implementation,

$$S^{(0)} = G(\pi_t, D_t, C_t). \tag{3}$$

The output is an oracle skill containing a bounded set of atomic operations. Each operation states a transferable procedure or decision criterion rather than a successful answer for one task. The resulting oracle skill is a deliberately strong and potentially redundant artifact. It may contain instructions that are useful only for the source task, duplicate operations that express the same principle, or assumptions that are valid in the reference implementation but not in the solver environment. Pruning therefore tests the behavioral necessity of each operation before the skill is generalized or placed in the runtime library. The authoring outputs and the host-side safeguards against leakage are described in Appendix A.2.

3.3 Verifier-Grounded Operation Pruning

We support two pruning modes. The first is a conservative leave-one-out procedure that serves as a fallback when the operation scores are unavailable or unreliable. The second uses an agent-produced removability score to reduce the number of expensive full solver evaluations. Both modes share the same acceptance rule: a deletion is eligible only when removing it is non-inferior to retaining it and the resulting skill remains above the no-skill floor,

$$\Delta_i(S^{(0)}, \mathcal{T}) \geq -z\,\mathrm{SE}_i \quad \text{and} \quad U_{\mathcal{T}}(S_{-i}) > U_{\mathcal{T}}(\varnothing), \tag{4}$$

where $S_{-i} = S^{(0)} \backslash \{o_i\}$, SE_i is the paired standard error of the full-versus-dropped comparison, and z is the configured pruning margin. A candidate that fails the verifier gate, falls below the no-skill floor, or is inconclusive remains unchanged.

Leave-one-out pruning. Starting from $S^{(0)}$, the harness renders a dropped variant for every reasoning operation and evaluates all variants, together with the full skill and the no-skill baseline, on the same common set of tasks, applying Eq. equation 4 to each. Every deletion is thus tested against the same task verifier, and a failed deletion leaves the source skill intact.

Score-guided block pruning. To reduce the number of expensive solver sweeps, a scoring agent assigns each reasoning operation a removability score, which serves as an ordering signal rather than evidence of correctness. Operations are sorted by score and partitioned into blocks; each block is deleted jointly and committed only if the reduced skill passes the same gate. A failed block is rolled back and recursively split until each remaining candidate is tested individually (Appendix A.3).

The final retained artifact is $S^\dagger = S^{(0)} \setminus \mathcal{D}$, where $\mathcal{D}$ is the set of accepted deletions. We call $S^\dagger$ a *verifier-screened core candidate*. It is not claimed to be globally minimal: the procedure uses greedy ordering, block rollback, and leave-one-out evidence rather than an exhaustive search over all operation subsets.

3.4 Skill Generalization

The generalizer G_{gen} receives the screened core candidate and the derivations that motivated its reasoning operations, but not the target task. It identifies the principle that makes each surviving operation effective, merges operations that express the same principle without introducing new ones, removes instance-specific anchors while preserving transferable procedures, and appends an explicit applicability section (Appendix A.4). Let $\tilde{S}$ denote the generalized version of $S^\dagger$. After structural validation, a behavioral gate evaluates $\tilde{S}$ on a held-out task group $\mathcal{G}$, disjoint from the source training split, and accepts the rewrite only when

$$U_{\mathcal{G}}(\tilde{S}) \geq U_{\mathcal{G}}(S^\dagger) - z_{\text{gen}}\,\mathrm{SE}_{\text{gen}} \quad \text{and} \quad U_{\mathcal{G}}(\tilde{S}) > U_{\mathcal{G}}(\varnothing). \tag{5}$$

If the gate fails, deployment rolls back to $S^{\dagger}$ rather than replacing a verified source skill with an unverified rewrite.

3.5 Runtime Skill Selection

Let $\mathcal{L} = \{S_s : s \in \mathcal{S}\}$ be the available skill library and let t be a target task. For each candidate skill S_s, the selector draws on four kinds of evidence. The *no-skill baseline* $b_t = U_t(\varnothing)$ measures how well the model performs on t without any injected skill; targets with $b_t > 0.80$ are gated immediately because the expected marginal gain does not justify the added context. The *direct transfer delta* $\delta_{s\to t} = r_{s\to t} - b_t$ is the empirically observed improvement when S_s was previously evaluated on t itself, and it enters the lookup table only after passing a positive-transfer filter. When no direct observation exists, *task similarity*

$$\rho(t,\, t') \;=\; \cos\big(\phi(t),\, \phi(t')\big) \tag{6}$$

is used to propagate evidence from the most similar previously evaluated task t', where $\phi(\cdot)$ is a fixed sentence embedding of the task description. Finally, *source quality* $q_{\text{src}}(s)$ is a shrinkage estimate of S_s's average transfer delta across all tasks on which it has been evaluated:

$$q_{\text{src}}(s) \;=\; \frac{n_s}{n_s+\kappa}\,\bar{\delta}_s \;+\; \frac{\kappa}{n_s+\kappa}\,\bar{\delta}_{\text{global}}, \tag{7}$$

where n_s is the number of observed evaluations, $\bar{\delta}_s$ the empirical mean delta for S_s, $\bar{\delta}_{\text{global}}$ the global prior mean, and κ a shrinkage constant (see Appendix A.5). A compatibility tier $c(s,t) \in \{\text{same, compatible, unknown, incompatible}\}$ encodes whether the source and target task types are structurally aligned; incompatible pairs are rejected by construction via a large negative bonus $\beta(c)$.

Selection proceeds in three phases of decreasing evidential strength. *Phase 1* uses direct transfer: the score $Q_{P1}(s,t) = \delta_{s\to t} + \beta(c(s,t))$ is accepted when positive. *Phase 2* propagates the best observed delta from the nearest neighbor $t^{\star} = \arg\max_{t'} \rho(t,t')$ (when $\rho(t,t^{\star}) > 0.35$): the score $Q_{P2}(s,t) = \delta_{s\to t^{\star}}\,\rho(t,t^{\star}) + \beta(c(s,t))$ is accepted when it exceeds a compatibility-dependent threshold $\tau(c)$. *Phase 3* is an empirical fallback combining the baseline-to-delta isotonic fit $\hat{f}(b_t)$ with source quality: $Q_{P3}(s,t) = \hat{f}(b_t) + q_{\text{src}}(s) + \beta(c(s,t))$. The runtime decision is

$$\sigma(t,\mathcal{L}) = \begin{cases} \varnothing, & b_t > 0.80, \\ s_{P1}, & Q_{P1}(s_{P1},t) > 0, \\ s_{P2}, & Q_{P2}(s_{P2},t) \geq \tau(c), \\ s_{P3}, & Q_{P3}(s_{P3},t) \geq \tau(c), \\ \varnothing, & \text{otherwise.} \end{cases} \tag{8}$$

The selector records the chosen source, compatibility tier, score, and confidence so that each runtime skill injection can be audited. Threshold values for $\tau(c)$ and bonus values for $\beta(c)$ are tabulated in Appendix A.5.

3.6 Evaluation and Evidence Boundary

Every pruning, generalization, and selection decision is evaluated through the same source-native harness, which isolates public task files from the reference implementation, reference outputs, and private judge (Appendix A.6). Binary BioDSBench scores and continuous BioMNIBench rewards are kept separate because they use different judges and task subsets. Across all protocols, a timeout or infrastructure failure is recorded as inconclusive rather than as evidence that a skill or operation is unnecessary. This evidence boundary prevents partial runs, incompatible denominators, or unverified generalized skills from being presented as measurements of the complete lifecycle.

4 Experiments

4.1 Experimental Setup

We evaluate our method using DeepSeek-V4-Flash (Xu et al., 2026) as the underlying agent model throughout the experiments. We intentionally use this relatively lightweight open-source model

Task type	No Skill	SkillOpt	Ours	Δ(Ours−NoSkill)
Survival analysis	0.73	1.00	**1.00**	+0.27
Association testing	0.78	0.85	**0.98**	+0.20
Mutation analysis	0.85	0.82	**1.00**	+0.15
Chromatin profiling	0.67	0.71	**0.93**	+0.26
Predictive modeling	0.68	0.62	**0.92**	+0.24
Differential expression	0.85	0.46	**0.98**	+0.13
Multi-omic integration	0.95	0.67	**1.00**	+0.05

Table 1: Performance comparison across biomedical scientific task types on BioMNIBench.AUTOSKILLS consistently outperforms the no-skill and SkillOpt baselines, achieving substantial gains in association testing, chromatin profiling, predictive modeling, and survival analysis, while reaching near-perfect or perfect performance on most task types.

Task type	No Skill	SkillOpt	Ours	Δ(Ours − No Skill)
Pathway enrichment	6/10	8/10	**10/10**	**+0.40**
Expression analysis	19/21	18/21	**21/21**	**+0.10**
Survival analysis	2/3	2/3	**3/3**	**+0.33**
Clustering	3/4	2/4	**4/4**	**+0.25**
Cross-cohort comparison	0/1	**1/1**	**1/1**	**+1.00**
All tasks	30/39 (76.9%)	31/39 (79.5%)	**39/39 (100%)**	**+0.23**

Table 2: BioDSBench results by task type. We reache full pass rate on the 39-task panel, while SkillOpt provides limited but positive cross-benchmark transfer.

rather than a stronger proprietary model, since the goal is to test whether verifier-grounded skill learning can improve the behavior of a model with limited baseline capability. This setting also allows the effect of the learned skills to be isolated: the no-skill and skill-enabled conditions use the same model, task protocol, and evaluator, with skill transfer being the primary intervention.

Our evaluation emphasizes scientific-data workflows, where success depends not only on producing a plausible final answer but also on following the correct data-processing and statistical procedure. We therefore use BioMNIBench(Qu et al., 2026) and BioDSBench(Wang et al., 2024a) as the main benchmarks. In contrast, conventional answer-oriented benchmarks generally do not expose the complete reference workflow and executable verification signals needed to evaluate whether an individual procedural skill is actually useful. We thus use LiveMathematician(He et al., 2026) and OfficeQA(Opsahl-Ong et al., 2026) only as auxiliary robustness checks rather than as the primary evidence for skill extraction.

4.2 MAIN RESULTS

BioMNIBench. BioMNIBench uses a graded reward in $[0, 1]$. We improves over the no-skill baseline, with the mean reward increasing from 0.790 to 0.917 (+0.127). Table 1 shows that the gains are distributed across diverse scientific task types rather than being concentrated in a single workflow family. In particular, chromatin profiling improves from 0.67 to 0.93 through peak-level stratified testing and same-type BH-FDR correction; association testing improves from 0.78 to 0.98 through stratified FDR correction; predictive modeling improves from 0.68 to 0.92 through feature-frequency analysis and validation; and clustering improves from 0.81 to 0.97 through cluster/factor–phenotype alignment. The remaining task types show the same pattern, with each gain corresponding to a concrete procedural correction rather than a generic increase in prompting context.

BioDSBench. BioDSBench uses binary pass/fail evaluation. On the 39-task evaluation panel, the no-skill baseline achieves 76.9% (30/39), SkillOpt transferred from BioMNIBench achieves 79.5% (31/39), and our method achieves 100% (39/39).

The task-type breakdown provides a more informative view than the aggregate score alone. SkillOpt(Yang et al., 2026) provides its main benefit on pathway enrichment, where the transferred skill

Method	BioMNIBench	BioDSBench
Ours	**0.917**	**1.000**
Ours w/o Skill Selection	0.870	0.795

Table 3: Ablation of runtime skill selection. We compare our full framework with a variant that removes the target-aware skill selection module while keeping the learned skill library and solver fixed.

contains reusable gene-ID mapping and enrichment-call procedures, while also resolving the remaining data-wrangling failure. Thus, the three-way comparison separates generic cross-benchmark transfer from target-aware skill selection. To verify the scalability of the method, we conducted tests on the scale-up dataset: we reaches 94.97% on a larger 159 evaluated tasks, compared with 77.36% for the no-skill baseline.

As auxiliary robustness checks outside the biomedical setting, we also evaluate LiveMathematician and OfficeQA with the same DeepSeek-V4-Flash solver. LiveMathematician improves from 0.57 to 0.68, and OfficeQA from 0.33 to 0.60 after skill transfer. These results are not used to establish the verifier-grounded extraction criterion; instead, they test whether the resulting skill remains useful when the task domain and output format change substantially.

4.3 Ablation Study

The central ablation isolates the contribution of runtime skill selection. All other components and the learned skill library are kept fixed, while the selector is removed so that the target agent receives skills without the proposed selection step.

Table 3 shows whether the benefit comes from merely injecting more procedural context or from selecting a skill that is compatible with the target task. The existence of non-improving and degraded BioMNIBench cases, together with the timeout failure observed for one SkillOpt transfer on BioDSBench, suggests that indiscriminate skill injection can introduce irrelevant context or additional execution overhead. We therefore leave the final quantitative conclusion on selection to the completed ablation runs rather than extrapolating from the current results.

4.4 Case Study

Figure 2 provides a qualitative view of how transferred skills change agent behavior. The BioMNIBench example (da-9-7) requires selecting exactly three CyTOF features and four Olink analytes and applying BH-FDR separately within each treatment arm. Without skills, the agent selects four CyTOF features, performs FDR only on the pooled analysis, and fails to recover the significant Arm C2 association. The transferred skill explicitly aligns each hypothesis stratum with its testing scope and recovers the Arm C2 finding ($\rho = -0.627$, $q = 0.037$), improving the score from 0.69 to 1.00.

The LiveMathematician example illustrates a different failure mode. The no-skill agent derives the correct $2n$ bound but then introduces an unsupported "one more dimension" argument and selects $2n + 1$. The transferred skill instead compares each candidate against the tight target bound and retains the unique statement consistent with $2n$, turning the score from 0.00 to 1.00.

These examples complement the aggregate benchmark results by showing that the learned skill stores procedural constraints and verification heuristics rather than task-specific answers. In both cases, the transferred skill targets the local failure mode that prevented the no-skill agent from completing an otherwise largely correct solution.

5 Conclusion

AI for Science increasingly requires agents to execute complex, long-horizon scientific workflows rather than isolated question-answering tasks. In such workflows, successful executions may contain redundant, source-specific, or unnecessary procedures, and indiscriminately reusing them can introduce irrelevant context into later scientific tasks.

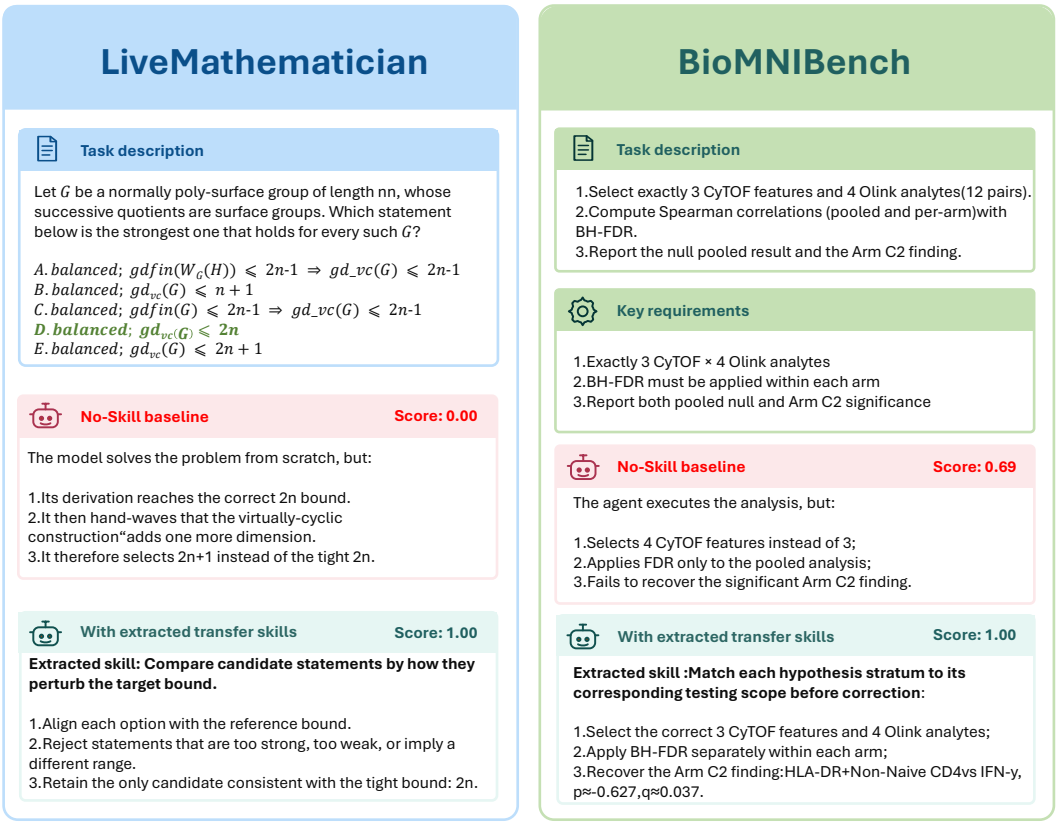

Figure 2: Case studies before and after skill transfer on BioMNIBench and LiveMathematic.

We presented AUTOSKILLS, a verifier-grounded framework for evolving reusable skills in long-horizon scientific workflows. Starting from a successful reference workflow, AUTOSKILLS constructs an over-complete skill and removes operations only when their deletion preserves task performance under the same solver and executable verifier. The surviving core is then generalized for transfer and selectively reused according to empirical evidence.

Our primary biomedical experiments demonstrate the practical value of this approach. On the reported BioDSBench panel, AUTOSKILLS achieves 100.0% accuracy, compared with 76.9% without skills and 79.5% with SkillOpt. On BioMNIBench, AUTOSKILLS achieves a mean reward of 0.917, while additional experiments on LiveMathematica and OfficeQA show transfer beyond the primary scientific setting. These results suggest that effective skill evolution can improve scientific task execution by retaining useful procedural knowledge while reducing unnecessary context.

More broadly, our work suggests that self-evolving scientific agents should not only accumulate successful procedures, but also verify which parts remain useful for future workflows. Executable evaluation provides a practical mechanism for this process, enabling agents to retain, generalize, and selectively reuse procedural knowledge as their experience grows.

AI USE STATEMENT

In this work, we did not use generative AI tools for research ideation, experimental design, methodology development, or writing the core content of this paper. We used generative AI tools only for minor grammar and language polishing of the manuscript text. We have reviewed all AI-assisted text to ensure it accurately reflects our original intent and findings. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

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.

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. Super: Evaluating agents on setting up and executing tasks from research repositories. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 12622–12645, 2024.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024.

Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. Livemathematicianbench: A live benchmark for mathematician-level reasoning with proof sketches. *arXiv preprint arXiv:2604.01754*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

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, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, et al. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026.

Yuanhao Qu, Yingzhou Lu, Xinming Tu, Serena Zhang, Tianwei She, Alexander Glenn Shaw, Jou-Ho Shih, Bingqing Zhao, Minjie Shen, Haochen Yang, et al. Biomnibench: Process-level evaluation of llm agents for real-world biomedical research. *bioRxiv*, pp. 2026–05, 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.

Zachary S Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. Corebench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363*, 2024.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.

Zifeng Wang, Benjamin Danek, Ziwei Yang, Zheng Chen, and Jimeng Sun. Can large language models replace data scientists in biomedical research? *arXiv preprint arXiv:2410.21591*, 2024a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

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*, 2026.

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.

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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

A Additional Method Details

This appendix provides the full specification of the components summarized in Section 3.

A.1 Multi-Round Harness

The harness evaluates a task through multiple judge-feedback rounds. At round r, the solver receives the public task, the rendered skill, and the compact feedback history $f_{t,i}$ from previous $r-1$ rounds,

$$h_{t,r} = (\pi_t, D_t, S, f_{t,1}, \ldots, f_{t,r-1}), \tag{9}$$

and produces a workspace state and candidate output $z_{t,r} = A_M(h_{t,r})$. The solver must call `finalize_submission` to indicate that its output is ready. The validator first checks the candidate against the public output contract, $a_{t,r} = H_t(z_{t,r})$.

If validation fails, the harness returns recoverable feedback to the same solver session; this event does not consume a judge round. If validation succeeds, the private judge is executed and returns a reward and feedback,

$$(y_{t,r}, f_{t,r}) = V_t(z_{t,r}), \qquad y_{t,r} \in [0, 1]. \tag{10}$$

For deterministic script judges, $y_{t,r} \in \{0, 1\}$; for rubric-based judges, $y_{t,r}$ is a bounded continuous reward. The number of judge attempts is capped by R. A run that times out or encounters an infrastructure failure is marked inconclusive and is never treated as a successful submission or a successful deletion.

A.2 Authoring Outputs and Leakage Safeguards

In addition to the atomic operations, the author agent also emits dependencies, operation roles, ablation scores, and owned resources. Host-side materialization checks the operation graph, rejects malformed anchors and unsafe paths, removes private-path tokens, and prevents reference files, judge data, and reference outputs from reaching the solver.

A.3 Full Pruning Procedure

Leave-one-out pruning. Starting from $S^{(0)}$, the harness renders a variant for every reasoning operation,

$$S_{-i} = S^{(0)} \setminus \{o_i\}, \tag{11}$$

and evaluates all variants, together with the full skill and the no-skill baseline, on the same common set of tasks under the acceptance condition of Eq. equation 4. Dependency references to a deleted operation are removed when the variant is rendered, while owned resources are removed together with their operation.

Leave-one-out pruning provides a simple safety guarantee: every deletion is tested against the same task verifier, and a failed deletion leaves the source skill intact. Its cost is high, however. With n operations, it requires a full multi-round solver sweep for the full skill, the no-skill baseline, and up to n dropped variants. Moreover, accepting each operation independently does not prove that the final combination of all accepted deletions will preserve accuracy, so the assembled skill is evaluated once more after pruning.

Score-guided block pruning. To reduce the number of expensive solver sweeps, the author or a dedicated scoring agent assigns each reasoning operation a removability score

$$p_i = G_{\text{score}}(o_i \mid \pi_t, D_t, C_t), \tag{12}$$

where a larger p_i means that the operation is more likely to be redundant and should be tried earlier. The score is an ordering signal rather than evidence of correctness. Operations are sorted in descending order of p_i and partitioned into blocks,

$$B_1, B_2, \ldots, B_K, \qquad \min_{o \in B_k} p(o) \geq \max_{o \in B_{k+1}} p(o). \tag{13}$$

At block B_k, the harness proposes $S' = S \setminus B_k$ and evaluates S' with the same solver, validator, and private judge. If S' passes the verifier gate, satisfies the non-inferiority condition, and remains above the no-skill floor, the whole block is committed and the next block is considered. If the first block attempt fails, the harness rolls back to the previous accepted skill S; none of the operations in the failed block is deleted. The failed block is then split into smaller score-ordered blocks. This recursive fallback continues until each remaining candidate is tested individually in descending score order. Thus, high-scoring operations receive the efficiency benefit of block deletion, while every ambiguous or failed block eventually falls back to the conservative single-operation test.

Plumbing operations. Plumbing operations are not scored by the single-item reasoning channel because they have no files or outputs to manipulate in that setting. They are carried through unchanged and checked in the full agent channel.

A.4 Generalization Details

To prevent the generalization step from becoming a second source of task-specific leakage, plumbing operations are withheld from the generalizer's prompt and spliced back into the output verbatim after rewriting.

Generalization follows four transformations. First, the agent identifies the principle that makes each surviving operation effective. Second, operations that express the same principle are merged; the number of reasoning operations may decrease or remain unchanged, but the generalizer may not introduce new operations. Third, instance-specific anchors are removed, including task names, author names, concrete file names, answer-specific constants, and symbols copied from one example. Transferable procedures, statistical tests, thresholds, normalization rules, validation checks, and ordering constraints are preserved. Finally, the agent appends an explicit applicability section that lists suitable task types, unsuitable task types, and caveats about assumptions or benchmark-construction regularities.

The generalized skill is structurally validated before deployment. Operation anchors must be well formed, dependencies must refer only to earlier operations, roles must be declared, and the rewritten skill must contain no private reference paths. Only after these checks does the behavioral gate of Eq. equation 5 evaluate the generalized skill on the held-out task group.

A.5 Runtime Selector Specification

The selector uses four kinds of evidence:

- the no-skill baseline $b_t = U_t(\varnothing)$;
- direct transfer evidence $\delta_{s\to t} = r_{s\to t} - b_t$, when source skill S_s has been evaluated on t;
- task similarity $\rho(t, t')$ between the target and a previously evaluated task;
- source quality $q_{\text{src}}(s)$ and a compatibility tier between the source and target task types.

Compatibility is represented by $c(s,t) \in \{\text{same}, \text{compatible}, \text{unknown}, \text{incompatible}\}$. The implementation uses the following bonuses and acceptance thresholds:

$c(s,t)$	same	compatible	unknown	incompatible
$\beta(c)$	$+0.15$	$+0.02$	0	-1.00
$\tau(c)$	-0.15	0.05	0.05	$+\infty$

(14)

Incompatible pairs are therefore rejected by construction. Targets with a sufficiently high no-skill baseline, $b_t > 0.80$, are gated and receive no skill because the expected gain is too small to justify additional context.

Phase 1: direct transfer. In the first phase, the selector uses a directly observed transfer whenever the source is available:

$$Q_{P1}(s,t) = \delta_{s\to t} + \beta(c(s,t)). \tag{15}$$

The direct source is returned when $Q_{P1}(s,t) > 0$. Direct observations enter the lookup table only after passing the configured positive-transfer filter.

Phase 2: nearest-neighbor transfer. If no accepted direct observation is available, the selector searches for a previously evaluated task t' most similar to t,

$$t^\star = \arg\max_{t'\in\mathcal{K}} \rho(t,t'), \tag{16}$$

where $\mathcal{K}$ is the set of tasks with observed positive transfer. When $\rho(t,t^\star) > 0.35$, the observed transfer is propagated with a similarity penalty:

$$Q_{P2}(s,t) = \delta_{s\to t^\star}\, \rho(t,t^\star) + \beta(c(s,t)). \tag{17}$$

The nearest-neighbor source is accepted when $Q_{P2}(s,t) \geq \tau(c(s,t))$. If this test fails, the selector falls back to the empirical score below and can override the failed nearest-neighbor choice with a stronger compatible candidate.

Phase 3: empirical fallback. For candidates without direct or nearest-neighbor evidence, the fallback score is

$$Q_{P3}(s,t) = \hat{f}(b_t) + q_{\text{src}}(s) + \beta(c(s,t)), \tag{18}$$

where $\hat{f}$ is a piecewise-linear interpolation of an isotonic fit from baseline performance to historical transfer deltas, and $q_{\text{src}}(s)$ is the shrinkage estimate of the source skill's average transfer quality. Candidates are ordered first by compatibility tier and then by $Q_{P3}(s,t)$. The top candidate is selected only if its score exceeds $\tau(c(s,t))$; otherwise the selector returns $\varnothing$ and loads no skill. Together, the three phases yield the runtime decision rule in Eq. equation 8.

A.6 Harness Isolation

The harness creates an isolated run directory, copies only public task files into the agent-visible workspace, and keeps the reference implementation, reference outputs, and private judge under a separate judge directory. A successful submission must pass public schema validation before the private judge is invoked. Judge feedback is compacted and returned to the same solver session for the next round, while raw trajectories and validation events are retained for auditing.

B Prompts

B.1 Harness / Solver System Prompt

Harness solver system prompt

```
\footnotesize
\begin{verbatim}
You are an autonomous solver inside a source-native evaluation harness
    .

Goal: produce valid final submission files for the current public task
    .

Use only run-local paths:
- public/: read-only public task data.
- workspace/: writable scratch, plans, and solver code.
- outputs/: writable final submission files.
- logs/agent/: optional notes.

Do not access private judge data, original task directories, other
    runs, parent
directories, home directories, or system secrets.

Use the provided Python runtime. Do not install packages or create/
    repair
environments.

Planning discipline:
- You may inspect necessary public files before planning.
- For each judge round, once you understand the task and any judge
    feedback, write
  a detailed, task-specific plan before implementation, long
    computation, or
  submission.
- Write the round plan to workspace/plans/round_NN.md and copy the
    current round
  plan to workspace/plan.md.
- The plan should be as detailed as needed: include task understanding
    , public data
  observations, assumptions, algorithm strategy, validation strategy,
    risks, and
  concrete execution steps.
- Avoid generic filler. Every plan section should contain information
    specific to
  this task or this judge feedback.
- Do not use TodoWrite; plans must be file artifacts.

Round closure discipline:
- Judge feedback is more valuable than private speculation. Prefer
    submitting a
  valid best attempt over running open-ended experiments.
- In each judge round, run only a small set of focused experiments,
    then select the
  best solver, regenerate outputs/, validate the file format, and call
  finalize_submission.
- If you believe outputs/ already contains files matching the output
    contract,
  submit the best available valid output instead of continuing
    unbounded
  exploration.

Experiment discipline:
- Put non-trivial Python experiments in workspace/experiments/*.py or
    workspace/*.py.
- Do not put long Python programs in Bash python -c; Bash should run
    scripts or
```

```
  short validation commands.

When final output files exist under outputs/, call finalize_submission
    . A text
answer is not a submission.
\end{verbatim}
```

Harness initial task prompt (BioMniBench branch)

```
\footnotesize
\begin{verbatim}
<run_context>
task_id: {task_id}
cwd: run root
python: {python}
max_judge_rounds: {max_rounds}
submission_dir: outputs/
current_plan_file: workspace/plan.md
round_plan_file: workspace/plans/round_01.md
</run_context>

<public_files>
{formatted public file listing}
</public_files>

<task_statement>
{README.md content, or: (No README.md content was found; inspect
    public/.)}
</task_statement>

<output_contract>
This is a BioMniBench data analysis task. The judge is an LLM that
    scores
your work against a rubric. You MUST produce exactly two files:
  - outputs/trace.md      Detailed analysis trace: code you ran,
    intermediate
                          results, plots described in text, reasoning.
  - outputs/answer.txt    Concise final answer to the task question.
Both files are evaluated by an LLM judge against a hidden rubric. Be
    specific,
show numerical results, and reference the data you actually computed.
</output_contract>

<visible_cases>
No discrete cases. Data files are at public/<filename> (symlinked from
public/data/), and public/visible_data/.
Read public/README.md for the full task description and data
    dictionary.
</visible_cases>

{active_skills block, if the Skill tool is enabled}

<workflow>
1. Read public/README.md carefully for the task question, data files,
    and any
   constraints.
2. Inspect public/data/ and public/visible_data/ to understand the
    actual file
   formats.
3. Write a detailed task-specific plan to workspace/plans/round_01.md
    and refresh
```

```
   workspace/plan.md.
4. Write Python solver code under workspace/ or workspace/experiments
    /.
5. IMPORTANT: cwd is run root. Data files are now symlinked directly
    at
   ./<filename> (e.g. ./BRCA_mRNA_top.csv). Also available at public/<
    filename>.
   Use public/data/ for subdirectory data.
6. The shared Python env already has pandas, numpy, scipy, scanpy,
    anndata,
   sklearn, statsmodels, matplotlib, seaborn, openpyxl. Do NOT install
     packages.
7. After computing your answer, write outputs/trace.md (analysis trace
    ) and
   outputs/answer.txt (final answer).
8. Call finalize_submission. The judge will score against a rubric.
</workflow>
\end{verbatim}
```

Harness initial task prompt (schema/cases branch)

```
\footnotesize
\begin{verbatim}
<run_context>
task_id: {task_id}
cwd: run root
python: {python}
max_judge_rounds: {max_rounds}
submission_dir: outputs/
current_plan_file: workspace/plan.md
round_plan_file: workspace/plans/round_01.md
</run_context>

<public_files>
{formatted public file listing}
</public_files>

<task_statement>
{README.md content, or: (No README.md content was found; inspect
    public/.)}
</task_statement>

<output_contract>
{output contract built from output_schema.json}
</output_contract>

<visible_cases>
{visible cases built from visible_data/cases.json}
</visible_cases>

{active_skills block, if the Skill tool is enabled}

<workflow>
1. Use <output_contract> and <visible_cases> as the submission
    contract. Inspect
   relevant public case input files for raw keys, shapes, dtypes,
    finite status,
   and value ranges before choosing a solver; do not assume array
    structure.
2. Once you understand the task, write workspace/plans/round_01.md and
     refresh
```

```
   workspace/plan.md.
3. Write solver code under workspace/ and longer experiments under
   workspace/experiments/.
4. IMPORTANT: Your cwd is run root. Always use paths relative to run
    root. Data
   files are at public/workdir/, NOT ../public/workdir/. When writing
    code in
   workspace/, use public/workdir/ for data paths.
5. Use Bash for short commands or to run scripts; do not use long
    inline python -c
   programs.
6. Run a bounded set of focused experiments, then choose the best
    current solver.
7. Write final submission files under outputs/ and run a quick local
    format check
   against <output_contract>.
8. Call finalize_submission. The judge feedback is more valuable than
    private
   speculation.
</workflow>
\end{verbatim}
```

Harness judge-feedback follow-up prompt

```
\footnotesize
\begin{verbatim}
<judge_feedback>
round: {round}/{max_rounds}
status: {status}
reward: {reward}
format: {format}
reason: {reasons joined by |, or none}
failed_metrics:
- {failed metric}
passed_metrics:
- {passed metric}
message: {judge message}
</judge_feedback>

{active_skills block, if the Skill tool is enabled}

<workflow>
Before modifying code for this feedback, understand the feedback, then
     write
workspace/plans/round_NN.md and refresh workspace/plan.md.
Make one focused fix or a small bounded experiment set, validate again
    , regenerate
outputs, then submit the best current output to the judge with
    finalize_submission.
Before finalize_submission, revalidate outputs against the same
    contract.
Use workspace/experiments/*.py for longer probes; do not put long
    Python programs
in Bash python -c.
</workflow>
\end{verbatim}
```

Harness no-finalize recovery prompt

```
\footnotesize
\begin{verbatim}
<no_finalize_recovery>
round: {round}/{max_rounds}
Your previous turn ended without finalize_submission.
Do not start new open-ended research or long experiments.
If you believe outputs/ contains valid final files, call
    finalize_submission now
with a concise summary.
If outputs/ is missing or invalid, make only the shortest necessary
    format fix,
then call finalize_submission.
If you cannot create a valid output, briefly explain the blocker and
    stop.
</no_finalize_recovery>
\end{verbatim}
```

Harness active-skills prompt block

```
\footnotesize
\begin{verbatim}
<active_skills>
The native Skill tool is enabled for this run.
At the start of each judge round, before writing the round plan or
    solver code,
call the Skill tool to inspect relevant active skills and incorporate
    any
applicable reusable guidance.
Use retrieved skill guidance as abstract process advice only; do not
    infer or
request hidden task materials from it.
In the round plan, include a section named Applied skills checklist
    before
implementation.
That checklist must name each applicable skill, list the abstract
    items you will
apply, state the anti-patterns you are avoiding, and define the cheap
    probe, stop
condition, expected runtime, and log path before any long run.
Before finalize_submission, write workspace/skill_application.json
    with
schema_version: 1 and a skills array.
Each exposed skill entry must include skill, status, evidence_path,
    and reason;
status must be used, not_applicable, or blocked_but_overridden.
Use evidence_path for a run-local public/workspace/logs artifact that
    proves the
contract item was checked; do not put private paths, reference-
    solution paths,
hidden data, or task-specific leaked constants in this file.
Allowed active skill names:
- {skill name}
</active_skills>
\end{verbatim}
```

Rollout system prompt (skillopt pipeline)

```
\footnotesize
\begin{verbatim}
You are a skilled bioinformatics research scientist working on a
    computational
biology task.

{skill_section}
## Task Format
You will be given a task description with data files to analyze. Use
    the available
tools (read files, write scripts, execute code) to complete the task.
    The harness
will evaluate your final answer against a rubric.

## Guidelines
- Read the task README carefully to understand the question and data
- Use bash commands to explore and process data
- Write Python/R scripts for analysis when needed
- Produce a final answer in the required format
- Think step by step before acting
\end{verbatim}
```

B.2 Skill Generalization Prompts

Skill generalization system prompt (BioMniBench / R

```
\footnotesize
\begin{verbatim}
You are an expert bioinformatics skill author. Given a **source task's
     oracle
skill** (a structured set of domain knowledge operations), you must
    produce a
**generalized, task-agnostic skill** that captures the core analytical
     pattern.

## Core Principle

You are given **only one task's skill**. You must NOT assume any
    specific target
task. Instead, **abstract the domain knowledge** so it can be applied
    to other
similar bioinformatics problems.

## What to Do: 4-Step Abstraction

### Step 1: Identify the Core Pattern
- What kind of analysis is this? (e.g., differential expression, GWAS,
     ChIP-seq
  peak calling, survival analysis, clustering, cell composition)
- What is the **statistical or computational method** at the heart of
    the analysis?
- What **data types** does it operate on? (e.g., gene expression
    matrices, peak
  files, variant calls, clinical tables)

### Step 2: Abstract the Operations
- Each operation should describe a **general analytical step**, not
    the specific
```

```
  task
- Replace concrete file names with generic placeholders
- Keep the **workflow structure** -- order and dependency between
    operations

### Step 3: Preserve Transferable Methods
- **KEEP**: Statistical tests, thresholds, normalization methods,
    validation
  criteria
- **KEEP**: Rubric structure, evaluation criteria, quality checks
- **REMOVE**: Specific dataset names, gene lists, disease names, exact
     column names
- **REMOVE**: Task-specific biological context that doesn't generalize

### Step 4: Declare Applicability Scope
This is CRITICAL. After the operations, you MUST include an `##
    Applicability`
section.

## Output Format

Use the same oracle-skill format with `<!-- ORACLE_OP_START op_XXX
    -->` and
`<!-- ORACLE_OP_END op_XXX -->` anchors.

```yaml
---
name: generalized-{source_task}
description: Generalized skill for {analysis_pattern} -- useful for
    similar tasks
  involving {domain}
source_task: {source_task}
source_task_type: {task_type if known, else unknown}
---

# Generalized Skill: {Abstract Title}

## 1. Operation Name

<!-- ORACLE_OP_START op_XXX_name -->
{generalized description of what this operation does, what methods to
    use, what to
watch out for}
<!-- ORACLE_OP_END op_XXX_name -->
```

Rules
- 8-12 operations, matching the source's structure
- Each operation must be self-contained and independently useful
- Use concrete, transferable methods (statistical tests, thresholds,
    normalization)
- Only use generic placeholders for data-specific details
- Do NOT reference any specific task name, file, or dataset

Applicability Section Requirements

After the operations, you MUST include this section. Be honest and
    precise:

```markdown
## Applicability

### Suitable For
```
```

```
- Task types: [list of task_type values this skill is appropriate for,
    e.g.,
  differential-expression, chromatin-profiling]
- Analysis patterns: [description of the analytical pattern, e.g.,
    comparing two
  groups of samples to find differentially abundant features]
- Data types: [what kind of input data this skill expects]

### NOT Suitable For
- Task types: [list of task_type values this skill is NOT appropriate
    for]
- Why: [brief explanation of the mismatch]
- Examples: [concrete counterexamples of tasks that would be misled by
    this skill]

### Caveats
- [Any limitations, assumptions, or conditions where this skill may be
    misleading]
- [Known failure modes: e.g., This skill assumes normally distributed
    data -- use
  with caution for count data]
```

Examples of Abstraction

Specific (bad)	Generalized (good)
Load `druggability_genes.csv`	Load the target gene/protein list (
CSV
format)
Filter to p-value < 0.05	Apply significance threshold (typical
p-value < 0.05)
Run Fisher's exact test on 2x2 contingency table	Keep -- this is a
transferable method
Extract ESR1 LBD mutations from column 7	Extract mutation status
from the
relevant genomic column
Compare CBFb-SMMHC vs Empty Vector	Compare treatment vs control
conditions
Use MACS2 with q<0.05	Run peak caller with appropriate
significance
threshold

Critical Warning

A skill that is **too specific** will mislead the agent when applied
    to a different
task.
A skill that is **too vague** will be useless.
The goal is to be **just specific enough** about methods while being
    **fully
generic** about the domain.

If the source skill is highly task-specific (e.g., analyze CBFb-SMMHC
    ChIP-seq
data), the generalized version should describe the **general ChIP-seq
    analysis
workflow** without mentioning CBFb-SMMHC or any specific cell line.
\end{verbatim}

Skill generalization system prompt (OfficeQA)

```
\footnotesize
\begin{verbatim}
You are an expert skill author for DOCUMENT QUESTION ANSWERING over
    office/government
documents (OfficeQA). Given a **source task's oracle skill** (a
    structured set of
operations that solve one specific question from an office document),
    you must
produce a **generalized, task-agnostic skill** that captures the core
    reasoning
pattern so it can transfer to other, similar document-QA questions.

CRITICAL: You receive ONLY this one task's skill. Do NOT assume any
    specific target
question. Abstract the procedure so it applies to a whole CLASS of
    similar
questions.

## Domain Context (OfficeQA)
These tasks answer questions from long office/government documents --
    Treasury
Bulletins, statistical yearbooks, budget reports, financial statements
    , agency
tables. Typical sub-skills:
- Locating the right table or section in a long document (via table of
     contents,
  section headings, table titles).
- Reading numeric cells correctly (right row + column, respecting
    units, scale,
  footnotes, and in millions/thousands headers).
- Multi-hop arithmetic: differences, sums, ratios, percentage change,
    growth rates,
  share-of-total.
- Unit & scale handling: millions vs billions, percent vs fraction,
    currency,
  rounding to the requested precision.
- Inflation / index adjustment (e.g. CPI-U) and other constant lookups
    .
- Cross-referencing multiple tables or an external constant.

## 4-Step Abstraction
### Step 1: Identify the Core Question Pattern
- What CLASS of question is this? (e.g. month-over-month change in a
    monetary
  aggregate, year-over-year growth rate of a line item, share of a
    category
  within a total, inflation-adjusted change).
- What is the reasoning/arithmetic at its heart?
- What document structures does it rely on (which kind of table/
    section)?

### Step 2: Abstract the Operations
- Each operation = a GENERAL step, not the specific question.
- Replace the specific table name, document, dates, and figures with
    generic
  descriptions (the target table, the reporting period, the prior
    period).
- KEEP the workflow order and dependencies between steps.

### Step 3: Preserve Transferable Method, Remove Specifics
```

```
- KEEP: the arithmetic formula (as a general formula), unit-handling
    rules,
  rounding rules, how to locate a table, how to handle footnotes,
    validation checks
  (sanity of magnitude/sign).
- REMOVE: the specific document name, specific table title, specific
    dates, specific
  numeric values/answers, specific series codes tied only to this
    question.
- If a constant is genuinely reusable (e.g. use CPI-U series to adjust
     for
  inflation), keep it as a METHOD (obtain the relevant CPI-U index
    values for the
  two periods), not as the specific 1979 numbers.

### Step 4: Declare Applicability Scope (CRITICAL)
After the operations, include an `## Applicability` section (see
    format below).

## Output Format
Use the same oracle-skill format with `<!-- ORACLE_OP_START op_XXX
    -->` and
`<!-- ORACLE_OP_END op_XXX -->` anchors.

```
---
name: generalized-{source_task}
description: Generalized OfficeQA skill for {question_pattern} --
    useful for similar
  document-QA questions involving {structure}
source_task: {source_task}
source_task_type: {task_type if known, else unknown}
---

# Generalized Skill: {Abstract Title}

## Overview
{one paragraph: what class of question this solves and the general
    approach}

## Operations

<!-- ORACLE_OP_START op_XXX_name -->
{generalized description of the step: what to locate/compute, what
    method/formula,
what to watch out for (units, footnotes, rounding)}
<!-- ORACLE_OP_END op_XXX_name -->
```

Rules
- Keep roughly the same number of operations as the source (do NOT pad
).
- Each operation self-contained and independently useful.
- Use concrete, transferable METHODS (formulas, unit rules, table-
    location tactics).
- Only generic placeholders for document-specific details.
- Do NOT reference the specific task name, document, table title,
    dates, or the
  final numeric answer.

Applicability Section Requirements
```markdown
## Applicability
```
```

```
### Suitable For
- Question patterns: [e.g. period-over-period change of a monetary/
    statistical
  aggregate, growth-rate / share-of-total computations from a document
     table]
- Document structures: [e.g. single time-series table with periods as
    rows and a
  value column]
- Arithmetic: [e.g. difference, ratio, percentage change, inflation
    adjustment]

### NOT Suitable For
- Question patterns: [patterns this skill does NOT fit]
- Why: [brief mismatch explanation]

### Caveats
- [units/scale assumptions, rounding, footnote handling, when the
    table-location
  tactic fails]
```

```
## Examples of Abstraction
| Specific (bad) | Generalized (good) |
|---|---|
| Find table MS-1 Currency and Coin in Circulation | Locate the target
  time-series table via the table of contents / section heading |
| May 1979 CPI-U = 69.4, June = 69.7 | Obtain the relevant CPI-U index
     values for
  the two comparison periods |
| Subtract real May value from June value = -1234.56 million | Compute
     the
  period-over-period change: current-period value minus inflation-
    adjusted
  prior-period value; keep result in the stated units |
| Round to nearest hundredth = 3.14 | Keep the rounding RULE: round to
     the
  precision requested by the question |

## Critical Warning
Too specific -> misleads on a different question. Too vague -> useless
    . Be just
specific enough about METHOD (formulas, unit/rounding rules, table-
    location tactics)
while fully generic about the document and figures.
\end{verbatim}
```

Skill generalization prompt (library-level

```
\footnotesize
\begin{verbatim}
[system]
Rewrite reusable agent skills. Never include target answers.

[user]
Generalize this skill for unseen tasks of type '{target_type}'.
    Preserve workflow,
checks, and output contracts; remove task-specific leakage.

{skill}
\end{verbatim}
```

B.3 SKILL EXTRACTION / ORACLE-AUTHORING PROMPTS

Oracle skill authoring system prompt (BioMniBench / R decomposition)

````
\footnotesize
\begin{verbatim}
You are an expert bioinformatics skill author. Given a **reference
    answer** (R code)
for a bioinformatics task, you must decompose it into a **structured
    oracle skill**
with 8-12 operations.

## What is an Oracle Skill?

An oracle skill is a structured set of domain knowledge operations
    that, when
followed step-by-step, guides an AI agent to solve a specific task
    correctly. Each
operation is a self-contained instruction with:
1. A clear title and purpose
2. Detailed guidance on what to do, methods to use, and pitfalls to
    avoid
3. Exact R code snippets showing how to implement the step

## Output Format

You must output a complete SKILL.md file with YAML frontmatter and
    operation
sections.

```yaml
---
name: oracle-{task_id}
description: Oracle skill for {task_id} -- {brief description}
task_id: {task_id}
analysis_type: {analysis_type}
---
```

Each operation uses ORACLE_OP_START/END markers:

```markdown
<!-- ORACLE_OP_START op_010_operation_name -->
## 1. Operation Title

### Purpose
What this step accomplishes.

### Instructions
Detailed step-by-step guidance.

### R Code
```r
# Exact R code for this step
```

### Expected Output
What the output should look like.

### Common Pitfalls
- What to avoid
- Edge cases to handle
<!-- ORACLE_OP_END op_010_operation_name -->
````
````

```
```

Rules for Decomposition

1. **8-12 operations**: Cover the complete workflow from data loading to final
   output
2. **Each operation must be independently useful**: A researcher could follow just
   this step
3. **Include exact R code**: The code should be directly runnable (with generic
   paths)
4. **Replace hardcoded paths**: Use `/workdir/` instead of specific paths where
   appropriate
5. **Preserve all logic**: Statistical tests, thresholds, transformations must be
   exact
6. **Include test assertions**: Reference the test cases so the agent knows what to
   verify
7. **Document edge cases**: What if data is empty? What if no DEGs found?

Example Operation Structure

Operation	Typical Content
op_010_data_loading	Load CSV files, handle row names, check dimensions
op_020_data_transformation	Transpose, merge, reshape
op_030_exploratory_analysis	Summary statistics, distributions
op_040_statistical_testing	t-tests, fold change calculations
op_050_visualization	Volcano plots, heatmaps, pathway diagrams
op_060_pathway_analysis	GSEA, KEGG, GO enrichment
op_070_result_interpretation	Filtering significant results, ranking
op_080_validation	Verify against test cases, check assumptions
op_090_output	Save results, generate final files

Critical

- Output ONLY the SKILL.md content (with YAML frontmatter)
- Each operation must contain runnable R code
- The code must be compatible with the reference answer's logic
- Include all test cases from the task as validation criteria
\end{verbatim}
```

**Reference-code authoring system prompt (R)**

```
\footnotesize
\begin{verbatim}
You are an expert R bioinformatics programmer. Your task is to write correct,
complete R code that passes the given test cases.

Rules:
1. Write ONLY valid R code, no explanations
2. The code must be self-contained (loads data, processes, outputs)
3. Data files are in public/data/ relative to the working directory
```
```

```
4. Output must be saved to outputs/submission.r or the judge won't
    find it
5. The code will be tested by appending test cases (R assertions) and
    running with
   Rscript
6. All required variables must be defined in the global scope for the
    test cases to
   access
7. Use library() calls for any required R packages

For data merge tasks (q0, q3):
- Load expression data with read.csv(public/data/..., row.names=1)
- Load labels from CSV
- Transpose expression data (samples to rows)
- Save original (pre-transpose) data as expression_data_t
- Merge with labels
- The test cases check: ncol(merged), nrow(merged),
  nrow(expression_data_t) == nrow(labels)

For differential expression tasks (q1, q4):
- Load expression data (if needed)
- Compute differential expression analysis
- Save results as a data frame named 'results'
- The test case checks: ncol(results) == 5

For DEG filtering tasks (q2, q5):
- Load DEG results
- Filter for significant genes
- Save as DEGAll
- The test case checks: ncol(DEGAll) == 7 or 8

For GO enrichment tasks (q6, q7):
- Load DEG results
- Run clusterProfiler::enrichGO
- Save as 'go' object
- The test case checks: !is.null(go)

For GSEA tasks (q8):
...
\end{verbatim}
```

Oracle skill extraction system prompt (OfficeQA)

```
\footnotesize
\begin{verbatim}
You are an expert at document-based question answering and information
     extraction.
You reverse-engineer the precise methodology that a domain expert
    would use to
extract or compute the correct answer from source documents, then
    distill it into a
reusable, structured SKILL.
\end{verbatim}
```

Oracle skill extraction user template (OfficeQA)

```
\footnotesize
\begin{verbatim}
```

```
Below is an OfficeQA task. You are given the QUESTION, the SOURCE
    DOCUMENT excerpt,
and the VERIFIED GOLD ANSWER. Your job: reverse-engineer the exact
extraction/reasoning methodology that yields the gold answer, then
    write a
structured oracle SKILL.

## QUESTION
{question}

## SOURCE DOCUMENT(S) (excerpt, may be truncated)
{source}

## VERIFIED GOLD ANSWER
{answer}

## INSTRUCTIONS
Write a SKILL.md that captures HOW to arrive at the gold answer.
    Requirements:
1. YAML frontmatter with: name, description, task_type (one of: table-
    lookup,
   numeric-calculation, date-extraction, entity-extraction,
    aggregation, comparison,
   multi-hop-reasoning), difficulty: hard.
2. A short ## Overview describing the class of question.
3. A ## Operations section: 4-8 numbered operations, each wrapped with
     anchors
   <!-- ORACLE_OP_START op_NN --> ... <!-- ORACLE_OP_END op_NN --> so
    each is
   independently prunable.
   Each operation = one concrete methodological step (locate the right
     table/section,
   identify the correct row/column/cell, handle units/formatting,
    perform the
   calculation, validate against expectations, format the final answer
     per the output
   contract {answer: ...}).
4. A ## Validation section: how to sanity-check the answer (units,
    magnitude,
   format).
5. Do NOT hardcode the literal gold answer as the answer is X. Instead
     describe the
   GENERAL method (e.g. locate the row labeled 'National defense'
    under the
   fiscal-year expenditure table, read the value in the target-year
    column, apply the
   unit stated in the question). The skill must generalize to similar
    questions, not
   memorize this one.

Output ONLY the SKILL.md content (with YAML frontmatter). No
    commentary.
\end{verbatim}
```

Oracle skill extraction system prompt (SearchQA)

```
\footnotesize
\begin{verbatim}
You are an expert at retrieval-augmented question answering and
    evidence-based
```

```
reasoning. You reverse-engineer the precise methodology that a domain
    expert would
use to locate the correct answer inside a set of retrieved documents,
    then distill
it into a reusable, structured SKILL.
\end{verbatim}
```

Oracle skill extraction user template (SearchQA)

```
\footnotesize
\begin{verbatim}
Below is a SearchQA task (Jeopardy-style clue answering over retrieved
     documents).
You are given the CLUE, the retrieved SOURCE DOCUMENTS excerpt, and
    the VERIFIED GOLD
ANSWER(S). Your job: reverse-engineer the exact retrieval/reasoning
    methodology that
yields the gold answer, then write a structured oracle SKILL.

## CLUE
{question}

## SOURCE DOCUMENTS (excerpt, may be truncated; delimited by [DOC]/[
    TLE]/[PAR])
{source}

## VERIFIED GOLD ANSWER(S)
{answer}

## INSTRUCTIONS
Write a SKILL.md that captures HOW to arrive at the gold answer.
    Requirements:
1. YAML frontmatter with: name, description, task_type (one of: entity
    -identification,
   fact-lookup, definition-matching, numeric-fact, date-fact, place-
    identification,
   multi-hop-reasoning), difficulty: hard.
2. A short ## Overview describing the class of clue and the reasoning
    pattern.
3. A ## Operations section: 4-8 numbered operations, each wrapped with
     anchors
   <!-- ORACLE_OP_START op_NN --> ... <!-- ORACLE_OP_END op_NN --> so
    each is
   independently prunable.
   Each operation = one concrete methodological step (parse the clue
    for the key
   entity/constraint, scan [TLE] titles for the most relevant document
    , locate the
   [PAR] passage containing the matching fact, cross-check the
    constraint in the clue
   against the passage, disambiguate among candidates, extract the
    concise answer
   span, format the final answer per the output contract {answer:
    ...}).
4. A ## Validation section: how to sanity-check the answer (does it
    satisfy every
   constraint in the clue? is it the most specific form? is it a name/
    place/short
   phrase, not a sentence?).
5. Do NOT hardcode the literal gold answer as the answer is X. Instead
     describe the
```

```
   GENERAL method (e.g. identify the clue's defining attribute, search
   titles/passages for a subject matching that attribute, verify the
    date/category
   constraint, extract the proper noun). The skill must generalize to
    similar clues,
   not memorize this one.

Output ONLY the SKILL.md content (with YAML frontmatter). No
    commentary.
\end{verbatim}
```

Trajectory analyst prompt (successful trajectory)

```
\footnotesize
\begin{verbatim}
You are analyzing a successful agent trajectory on a BioMNIBench
    bioinformatics task.

The agent achieved a good score. Review the task description, actions,
     and feedback.

Identify the key success factors:
1. What methods and approaches worked well?
2. What tool usage patterns were effective?
3. What domain knowledge was correctly applied?
4. Could the approach be generalized into a reusable skill?

Provide a concise analysis.
\end{verbatim}
```

Trajectory analyst prompt (failed trajectory)

```
\footnotesize
\begin{verbatim}
You are analyzing a failed agent trajectory on a BioMNIBench
    bioinformatics task.

The agent attempted to solve a computational biology task but achieved
     a low score.
Review the task description, the agent's actions, and the evaluation
    feedback.

Identify the key failure points in the trajectory:
1. Did the agent misunderstand the task requirements?
2. Did the agent use incorrect methods or tools?
3. Did the agent fail to interpret results correctly?
4. What specific skill knowledge would have helped?

Provide a concise analysis and suggest concrete skill improvements.
\end{verbatim}
```

B.4 SKILL SELECTION / INJECTION PROMPTS

LLM skill selector prompt

```
\footnotesize
\begin{verbatim}
You are choosing at most ONE reusable analysis SKILL to help an
    autonomous coding
agent solve a bioinformatics R task BEFORE it starts.

## TASK
task_id: {tid}
title: {title}
task_type: {ttype}
data_files: {data files, or (none listed)}

## AVAILABLE SKILLS (each is a generalized reusable procedure)
[{i}] id={id} | type={source_task_type} | {name}
     {description}

## INSTRUCTIONS
Pick the single skill most likely to help this task succeed, OR pick
    NONE if no
skill genuinely fits (injecting an irrelevant skill can hurt).
A good skill matches the task's analytical pattern (task_type / method
    ), not just
the biological topic.
Respond with ONLY one line of JSON: {skill: <id or none>, reason: <
    short>}
\end{verbatim}
```

Skill-injection wrapper (ablation)

```
\footnotesize
\begin{verbatim}
{finalize preamble}
You are given a proven methodology skill for this class of task.
    Follow it to
produce the correct submission.

<skill>
{skill content}
</skill>
\end{verbatim}
```

Native Skill tool invocation prompt

```
\footnotesize
\begin{verbatim}
Execute a skill within the main conversation

When users ask you to perform tasks, check if any of the available
    skills match.
Skills provide specialized capabilities and domain knowledge.

When users reference a slash command or /<something> (e.g., /commit,
/review-pr), they are referring to a skill. Use this tool to invoke it
    .
```

```
How to invoke:
- Use this tool with the skill name and optional arguments
- Examples:
  - `skill: pdf` - invoke the pdf skill
  - `skill: commit, args: -m 'Fix bug'` - invoke with arguments
  - `skill: review-pr, args: 123` - invoke with arguments
  - `skill: ms-office-suite:pdf` - invoke using fully qualified name

Important:
- Available skills are listed in system-reminder messages in the
    conversation
- When a skill matches the user's request, this is a BLOCKING
    REQUIREMENT: invoke
  the relevant Skill tool BEFORE generating any other response about
    the task
- NEVER mention a skill without actually calling this tool
- Do not invoke a skill that is already running
- Do not use this tool for built-in CLI commands (like /help, /clear,
    etc.)
- If you see a <command-name> tag in the current conversation turn,
    the skill has
  ALREADY been loaded - follow the instructions directly instead of
    calling this
  tool again
\end{verbatim}
```

B.5 Skill Pruning (Lifecycle) Prompts

Skill pruning system prompt (R bioinformatics)

```
\footnotesize
\begin{verbatim}
You are an expert at pruning R bioinformatics skills for an AI coding
    agent.

Your job: Given a **full oracle skill** with N operations, produce a
    **pruned
version** that keeps ONLY the operations essential for passing the
    judge's test
cases.

## Judge Test Cases

The judge works by:
1. Reading the agent's submission.r file
2. Appending R assertions (test cases) to the code
3. Running with Rscript
4. If exit code == 0, score = 100; else score = 0

So the test cases check that **specific R variables exist with the
    correct
structure**.

## Pruning Principles

1. **KEEP operations that create variables checked by test cases** --
    These are
   non-negotiable
2. **KEEP operations that load required data** -- Data files must be
    loaded
```

```
3. **KEEP operations that install/load required libraries** --
    Libraries must be
   loaded
4. **REMOVE cosmetic operations** -- Like resetting row names,
    reordering columns,
   adding comments
5. **REMOVE redundant operations** -- If two ops can be combined,
    combine them
6. **Combine related operations** -- Data loading ops can be merged
7. **Target: 4-6 operations** -- Pruned but still complete enough to
    pass

## Output Format

You must output a complete pruned SKILL.md file. The format is:

```markdown
---
source_task: {task_id}
task_type: {category}
language: R
pruned: true
generated_by: oracle_pruning
verification: Pruned by LLM analysis
---

# Oracle Skill (Pruned)

## Task Context

[Task description]

## Verification Criteria

```r
[judge test case]
```

Operations

Operation 1

```r
# Code
```

Operation 2

```r
# Code
```


...
```
\end{verbatim}
```

Skill pruning system prompt (generic oracle skills)

```
\footnotesize
\begin{verbatim}
```

```
You are an expert at pruning oracle skills. Given a complete oracle
    skill with N
operations, your job is to create a **pruned version** that keeps ONLY
     the essential
operations needed to achieve 100% score on the task.

## Pruning Principles

1. **Remove redundant operations**: If two operations can be combined,
     combine them
2. **Remove overly verbose instructions**: Keep only the core logic
3. **Remove operations that don't affect the output**: Skip
    explanatory steps that
   don't change the code
4. **Keep ALL operations that produce required outputs**: Any step
    that creates a
   variable or file used in the final answer must stay
5. **Keep ALL validation-critical operations**: Steps that verify
    against test cases

## Output Format

Same format as the original oracle skill, but with fewer operations
    (4-7 operations,
ideally 5).
\end{verbatim}
```

B.6 LLM JUDGE / GRADER PROMPT

LLM judge / grader prompt (rubric-based)

```
\footnotesize
\begin{verbatim}
You are an expert evaluator for a data analysis task.

Evaluate the agent's work using the following rubric:

{rubric}

Here is the agent's analysis trace:

<trace>
{trace_content, or [No trace file provided]}
</trace>

Here is the agent's final answer:

<answer>
{answer_content, or [No answer file provided]}
</answer>

For each criterion in the rubric, choose ONE level: A, B, or C. Only
    output JSON with
the format:
{
  criteria: {
    criterion_1: {level: A, reasoning: ...},
    criterion_2: {level: B, reasoning: ...}
  },
  overall_reasoning: ...
}
```

```
\end{verbatim}
```

B.7 Auxiliary Prompts

Task-type classifier prompt

```
\footnotesize
\begin{verbatim}
You are a bioinformatics data-science task classifier. Classify the
    task below.

Choose exactly ONE task_type from this controlled vocabulary:
{task type vocabulary}

Choose exactly ONE category (biomedical domain) from:
{category vocabulary}

Task ID: {task_id}

--- TASK MATERIAL ---
{text}
--- END ---

Rules:
- task_type must be the single best analytical method/pattern the task
     performs.
- If the task is mostly data loading/reshaping/merging with no
    downstream stats, use
  data-wrangling.
- If it is a plain statistical hypothesis test, use statistical-
    testing.
- category is the biomedical domain of the data (guess from disease/
    tissue names; use
  general-biology if unclear).

Respond with ONLY a compact JSON object, no markdown, no explanation
    outside JSON:
{task_type: <one-of-vocabulary>, category: <one-of-categories>,
 rationale: <=20 words}
\end{verbatim}
```