When Tools Dominate, When Strategies Matter: A Causal Study of Agentic Computational Biology

Anonymous authors
Paper under double-blind review

Abstract

Agentic AI increasingly automates computational biology workflows, yet performance varies widely and inconsistently across benchmarks, and observational studies cannot separate tool heterogeneity from planning-strategy differences. We introduce Dissect-Audit, a causal-study framework that isolates the two via controlled factorial experiments on single-cell RNA-seq analysis, using a shared registry of validated operations to independently vary the tool layer and planning strategy under do-calculus logic. Across our 23-task core benchmark, tool-layer unification alone eliminates inter-strategy variance (Wilcoxon $p = 1.22 \times 10^{-4}$) for all eight frontier LLM planners tested: every planner gains quality and sheds run-to-run variance (+0.08 to +0.20, mean +0.15; variance reduced 42–100%), and planners sharply separated under native tools become statistically indistinguishable (Kruskal–Wallis $p = 0.61$). Added validation yields no further gain, marking validators as reactive rather than proactive. Boundary experiments show tool quality binds only for in-distribution tasks with a canonical workflow; on out-of-distribution tasks hinging on a methodological decision, inter-strategy variance persists ($0.125 \rightarrow 0.135$) and a runtime oracle revives the strategy advantage (+0.57) — so the governing boundary is oracle × IID/OOD, not bio-vs-general. We package these diagnoses into a reusable protocol whose transplant uplift self-targets lower-quality agents (corr = −0.86, $p = 0.006$).

1 Introduction

The application of LLM-based agents to computational biology has accelerated dramatically. Systems such as K-Dense (Li et al., 2025), Robin (Ghareeb et al., 2026), Biomni (Huang et al., 2026), Claude Science (Anthropic, 2026), OpenScience (Synthetic Sciences, 2026), and domain-specific agents for single-cell analysis (Xiao et al., 2026; Wu & Tang, 2025) have demonstrated the ability to autonomously execute complex bioinformatics pipelines (from quality control to biological interpretation) with minimal human intervention. Yet benchmark evaluations reveal a troubling pattern, where performance variance across agents is large and inconsistent. On BiomniBench-DA (Qu et al., 2026), process-level rubric scores vary by more than 20 points (out of 100) across agent–harness configurations on identical tasks, and the choice of harness alone can shift a fixed model's score by more points than a full model-generation upgrade, evidence that current agents' reliability depends as much on scaffolding as on underlying capability.

A bio-agent is a general LLM reasoning core wrapped in a domain-specific tool layer, so reasoning and planning paradigms such as ReAct (Yao et al., 2023b) and Tree-of-Thoughts (Yao et al., 2023a) transfer directly, and the harnesses we study (Appendix C.6) instantiate them. What does not transfer is the semantics of failure. A central general-agent result, that once tool access is standardized planning becomes the dominant bottleneck (Jimenez et al., 2024; Liu et al., 2024), was established under an implicit assumption: that correctness is observable, i.e. a runtime oracle exists. In a silent-failure domain with no such oracle, that assumption breaks. This motivates our central question: *when tool access is standardized but correctness is unobservable, is planning still the dominant bottleneck, or does the correctness of the tool implementations become the binding constraint?*

The conventional explanation for inter-agent variance is differences in tool availability: one agent may have a batch-correction tool but another lacks. This "tool gap" hypothesis is intuitive and partially supported (agents with more tools score higher on average (Qu et al., 2026)), but conflates correlation

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.

with causation. Modern bio-agents differ along at least four dimensions: **tool availability**, **planning strategy**, **verification logic**, and **prompt design**. Worse, "tools" itself conflates three factors prior work rarely separates: **availability**, **selection**, and **correctness** (is the invoked tool methodologically sound, or does it silently corrupt the analysis). General-agent benchmarks study the first two under the assumption that tools are correct-by-construction; in biology the dominant failure mode is the third: a tool available, correctly selected, and yet silently wrong.

Because these four dimensions co-vary across agents, observational benchmarks cannot decompose this variance: a superior score may reflect better tools, a better strategy, or both. What the field lacks is a controlled intervention: fix one factor (the tool layer) and vary another (the planning strategy) to measure each one's causal effect. This is analogous to mediation analysis (Baron & Kenny, 1986), decomposing an architecture's total effect into direct effects (strategy, verification) and indirect effects (mediated through tool quality). Without such decomposition, resource allocation in agent development is guided by intuition rather than evidence.

We introduce Dissect-Audit, a framework for factorial experimentation on bio-agent architectures. At its core is a shared skill registry of six validated sc-RNA-seq operations (Appendix Table 5) that can replace or supplement any agent's native tools. Around the registry we layer two components: an error-injection mode that simulates tool heterogeneity by stochastically introducing known failure modes at calibrated rates (e.g., double normalization; Appendix Table 6), and an output validation sub-layer that encodes domain-specific methodological knowledge as automated checks. A harness abstraction isolates planning strategy from tool execution, so planning algorithms can be swapped and compared freely. The design follows the do-calculus (Pearl, 2009): rather than merely observing agents that happen to use unified tools, we perform the intervention $do(\text{tool_layer} := \text{unified})$, forcibly assigning every agent the same tool layer. This removes the experimentally introduced tool-layer heterogeneity and exposes residual variance attributable to planning under the shared execution substrate. This work makes five contributions:

1. **Causal-experimental framework.** We introduce a factorial experimental design for bio-agents, separating tool-layer and strategy-layer effects rather than evaluating agents only as end-to-end systems.
2. **Real-data benchmark with ground-truth failure modes.** A 27-task benchmark spanning the full sc-RNA-seq analytical taxonomy, of which 21 follow a canonical analytical workflow and 6 require a methodological decision; each task carries known failure modes that make tool- and strategy-induced errors independently observable.
3. **Primary experimental finding.** Unifying the tool layer drives inter-strategy performance variance to zero in our error-injection setup, establishing that on canonical bio-analysis workflows the binding constraint is tool-implementation *correctness*, not tool availability and not planning sophistication.
4. **Cross-model LLM validation.** The effect is not a single-model artifact: across eight frontier LLM planners from six vendors, every model gains under unified tools, and between-model differences observed under the error-injected tool condition are substantially compressed and are no longer statistically significant once tools are unified.
5. **Boundary map and reusable audit protocol.** We map *when* tools versus strategy dominate along an **oracle** $\times$ **IID/OOD** axis, and package these diagnoses into Dissect-Audit, a transplant, preventive-gating, and minimal-sufficient-skill-set protocol whose uplift is largest for lower-performing planners in our controlled evaluation.

In short, on the workflow-specified scRNA-seq tasks studied here, tool reliability explains a substantial fraction of observed performance variance under simulated tool heterogeneity, while planning differences persist on decision-structured tasks and in settings where runtime feedback can be used for adaptation. Our evidence is confined to sc-RNA-seq under *simulated* tool heterogeneity; it is the methodological template, rather than the specific effect magnitude, that we expect may transfer.

2 Related Work

Agents for biological analysis. The past two years have witnessed a proliferation of LLM-based agents for computational biology, distinguished from earlier automation tools such as Galaxy workflows (Afgan et al., 2018) and Snakemake pipelines (Mölder et al., 2021) by their use of natural language to interpret tasks, select tools, and adapt execution dynamically. Biomni (Huang

et al., 2026) is a general-purpose biomedical agent that decouples the agent from its action space (Biomni-E1 exposes 150 biomedical tools, 105 software packages, and 59 databases; Biomni-A1 retrieves resources, refines plans, and executes workflows as code), illustrating how differences in accessible tools, resource selection, planning, and execution jointly drive performance variation. Domain-specific single-cell agents, including CellAgent (Xiao et al., 2026), scExtract (Wu & Tang, 2025), ChatGPT-based assistants (Hou & Ji, 2024), and collaborative multi-agent systems (Ghareeb et al., 2026), demonstrate competence but share a limitation: they are evaluated as monolithic systems, making it impossible to attribute success or failure to specific architectural components.

Agent architectures and planning paradigms. Our harnesses instantiate established patterns: ReAct (Yao et al., 2023b) (adaptive planner), Reflexion (Shinn et al., 2023) (verify-and-retry), and Chain-/Tree-of-Thought (Wei et al., 2022; Yao et al., 2023a) (implicit in the LLM planner, with prompting held constant). Toolformer (Schick et al., 2023) and Gorilla (Patil et al., 2024) target tool-*selection*; we instead isolate tool *quality* (what happens after the right tool is selected but its implementation introduces errors), and the Model Context Protocol (Anthropic, 2024) supplies the standardized invocation our MCP Bridge uses to serve external agents. Multi-agent scientific systems (Huang et al., 2026; Ghafarollahi & Buehler, 2025) compound the attribution problem, which our single-agent factorial design is built to resolve.

Benchmarks and evaluation. Existing agent benchmarks are almost uniformly observational. AgentBench (Liu et al., 2024) varies the LLM while holding tools fixed, the inverse of our design; ToolBench (Qin et al., 2024) and API-Bank (Li et al., 2023) measure tool-selection accuracy but not whether tools drive performance; SWE-bench (Jimenez et al., 2024) standardizes tools and attributes variance to reasoning, but without the causal framing; ScienceAgentBench (Chen et al., 2025) reports tool-availability correlations without testing causation; BLADE (Gu et al., 2024) scores agents' analytical decisions against expert ground truth but does not attribute outcomes to tool, planning, or verification layers; BioPlanner (O'Donoghue et al., 2023) scores plan correctness in isolation from tool execution; and BiomniBench, closest in scope, evaluates agents across biology modalities observationally, each carrying its own tools, prompts, and planning logic. Because all evaluate agents as black boxes, a score reflects the joint effect of tool availability, planning, verification, and prompt design. Dissect-Audit instead provides white-box causal evaluation: we decompose the agent into layers, intervene on each independently, and measure each layer's contribution.

Causal inference in ML evaluation. We adopt a controlled-experiment perspective from factorial design (Montgomery, 2020): the tool-layer condition (native vs. unified) is the treatment, task identity is a blocking variable, and inter-harness variance is the outcome. The target estimand is $P\big(\text{variance} \mid do(\text{tool_layer} = \text{unified})\big)$ (Pearl, 2009), distinct from merely conditioning on agents that happen to use unified tools. Viewed through mediation analysis (Baron & Kenny, 1986), strategy affects performance along `planning` → `invocations` → `tool quality` → `performance`; our Phase 1 result shows this mediated pathway dominates. This extends to bio-agents a controlled-ablation tradition (Qin et al., 2024; Sclar et al., 2024; Hutter et al., 2014) otherwise uncommon in agent evaluation.

3 Results

3.1 Design Overview

Each experimental run pairs a *planning strategy* with one of three *tool layers* (Appendix C.4): **native**, six sc-RNA-seq skills with calibrated stochastic error injection (e.g. double normalization; Appendix Table 6); **unified**, the same skills with clean, state-aware implementations; and **unified + validated**, which adds post-hoc validation of every output. Planning strategies comprise three rule-based harnesses (Appendix C.6): a fixed-order **linear pipeline**, a metadata-aware **adaptive planner**, and **verify-and-retry**, which validates after each step and re-runs failures, plus a stochastic **LLM planner** driven by frontier models. All execute through a single skill registry, so strategy and tool layer are independently manipulable. The native-like heterogeneity here is *simulated*: we inject documented failure modes at calibrated rates rather than sampling real APIs, which keeps the error-injected–unified comparison experimentally controlled but makes its *magnitude* calibration-dependent (Appendix G). Runs are scored by a composite of validator-severity counts with task-level ground truth where available (Appendix C.7); the benchmark, datasets, and run matrix are in Appendix B.

3.2 Tool Unification Collapses Inter-Strategy Variance on Canonical Workflows

The Phase 1 primary experiment tests whether holding the tool layer constant reduces inter-harness performance variance. We compare three rule-based planning strategies (linear pipeline, adaptive planner, verify-and-retry) across 23 real-data tasks under native versus unified tool layers, with 10 repeats per condition (total: 2,070 runs). Under native tools, the three strategies diverge: verify-and-retry partially recovers from injected errors while the other two cannot, which creates measurable inter-harness variance. Under unified tools, all three converge to identical output quality.

Tool unification drives inter-harness variance to zero: mean per-task variance falls from 0.0025 to zero, with 13 of 23 tasks (57%) showing non-zero native variance; the mean composite score rises from 0.852 to 0.970 (Appendix Table 15). Figure 1 visualizes this: per-task inter-harness variance collapses to zero under unified tools (panels A–B), and the residual native-tool spread is driven by the harnesses' differential error recovery (panel C). Appendix Figure 8 traces the same convergence at the level of individual tasks: harness score trajectories diverge under native tools and overlap under unified ones.

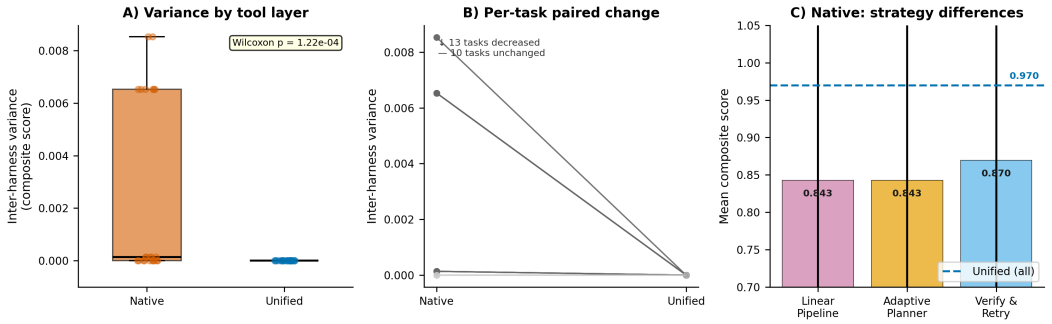

Figure 1: **Phase 1 inter-harness variance under native vs. unified tools (rule-based harnesses).** **(A)** Per-task inter-harness variance by tool layer (each point is one of the 23 tasks); native variance is right-skewed while unified is uniformly zero (Wilcoxon $p = 1.22 \times 10^{-4}$). **(B)** Paired change per task (each line is one task; dark = non-zero native variance, light = already zero); no task increases. **(C)** Mean composite score per harness under native tools, with the unified all-harness mean (dashed line) as reference.

We test the paired differences in inter-harness variance (paired by `task_id`) with two complementary tests. The parametric paired t-test gives $t = 3.447$, $p = 0.0023$, and the pre-registered one-sided Wilcoxon signed-rank test (native > unified) gives $W = 91.0$, $p = 1.22 \times 10^{-4}$ (matched-pairs rank-biserial $r_{rb} = 1.00$, as all 13 tasks with a non-zero native−unified difference favor unified). Both reject the null of no tool-layer effect on inter-harness variance at conventional significance levels.

The source of the native-mode variance is differential error-recovery ability (Appendix Table 8): under native tools verify-and-retry achieves a 22 pp higher pass rate (95.7% vs. 73.5%) by detecting and retrying failed steps, but this advantage vanishes under unified tools where all harnesses reach 100% pass rate and score 0.970. The linear pipeline and adaptive planner score identically (0.843): data-awareness cannot help when the *execution* of well-planned steps is corrupted. Variance concentrates in tasks with more operations or state-detection requirements (largest: `real_preproc_04` and `real_interp_04`, 0.0085 each, followed by a six-way tie at 0.0065; Appendix Table 9), which offer more opportunities both for errors to be injected and for harnesses to diverge: verify-and-retry recovers on the six tied tasks (1.0 vs. 0.86 for the rule-based plans) but exhausts its step budget on the two hardest, scoring 0.0 against their 0.16. Causally, do(tool_layer := unified) blocks the path from tool errors to outcome variance, leaving zero residual strategy variance: the full observed variance is tool-mediated.

Validator impact is an informative null. Adding active output validation on top of the unified layer (unified vs. unified+validated) yields no marginal benefit, as the zero-variance result above shows that the two conditions are numerically identical across all 690 runs per condition (mean 0.970, std 0.118, 100% pass, 0 errors, 420 warnings each), so no test is needed. The 420 warnings are exclusively batch-effect flags on multi-batch Kang2018 (Kang et al., 2018) tasks analyzed without

explicit integration, a methodological recommendation rather than an execution error. This null is informative, which means output validators are *reactive safeguards* whose value is conditional on tool unreliability (they detect the 132 native-mode errors but have nothing to catch on clean tools), not *proactive quality enhancers*. The design implication is to layer validators as defense-in-depth rather than rely on them as the primary quality mechanism, since investing in tool reliability beats investing in error detection.

3.3 Eight Frontier LLMs Reproduce the Tool-Layer Effect

Phase 3 replaces rule-based planning with a real LLM planner, the most realistic test of our hypothesis. The LLM generates genuinely different plans across repeats, so we can measure whether tool-layer unification reduces outcome variance even when planning is non-deterministic. To rule out single-model idiosyncrasies, we replicate the full protocol across eight frontier LLMs spanning six vendors. Each model plans the full 27-task panel under all three tool layers with 10 repeats (810 runs per model); to stay scope-matched to the 23-task Phase 1 benchmark, the analysis below is limited to those 23 tasks (690 per model), with the 4 OOD decision tasks held out and analyzed separately (§3.4 and §3.7). We present the reference model (Claude Opus 4.6) first, then the cross-model generalization; for the reference model, the pattern from Phases 1–2 reappears under genuinely stochastic planning (Appendix Table 14).

Tool-layer unification improves the LLM agent's mean score by **+26%** (0.734 → 0.928) and reduces run-to-run standard deviation by **44%** (0.347 → 0.193). The improvement is consistent across tests: a Mann–Whitney U test on scores (unified > native) gives $U = 39{,}930$, $p = 1.2 \times 10^{-23}$; a Wilcoxon signed-rank test on per-task variance (native > unified) gives $W = 156$, $p = 9.2 \times 10^{-4}$ across the 18/23 tasks with a non-zero difference; and the native-to-unified effect size is Cohen's $d = 0.69$ (medium-to-large). The unified versus unified+validated comparison is null ($U = 26{,}450$, $p = 0.50$), consistent with Phase 2.

The gains concentrate where tool errors are most damaging (Appendix Table 11): integration tasks improve most (+0.438, where `ignore_batch_in_hvg` at rate 0.80 is most damaging), interpretation next (+0.300), and feature engineering least (+0.047, HVG being robust to upstream errors). Integration remains the weakest category even under unified tools (0.70), genuine planning difficulty rather than a tool problem, confirming that tool unification *reveals* the residual orchestration bottleneck. Native run-to-run variance also scales with difficulty (easy 0.006, medium 0.006, hard 0.076, expert 0.004; Appendix Table 12), with unified tools largely eliminating it (only hard integration tasks retain a small residual, 0.012; the expert dip reflects a floor effect where those runs fail completely).

Across all planners (Appendix Table 10, Figures 7 and 11), the LLM planner benefits most from unification (+0.194 vs. +0.100–0.127 for rule-based) because its stochastic plans interact with native errors unpredictably, yet achieves the lowest unified score (0.928 vs. 0.970), a 4.3% gap quantifying the pure orchestration-quality difference that tool improvement alone cannot close. Across all eight models (Figure 2, Appendix Table 17, and Appendix Figure 9), **every model improves under tool unification**, from +0.046 to +0.217.

Three cross-model findings emerge. *The tool-unification benefit is consistent across all eight models.* All the models improve significantly (Mann–Whitney $p < 10^{-4}$ in every case, $p < 10^{-22}$ for six of the eight), with effect sizes from small to large (Cohen's $d = 0.17$–0.84); the detrimental effect of the injected tool failures is observed across all tested planners. DeepSeek-V4-Pro and Gemini-3.7-flash are the mild outliers ($d = 0.19$ and 0.17; $\Delta = +0.053$ and $+0.046$), but this reflects their already-strong native performance (0.843 and 0.849, the two highest of any model). They simply have less room to gain, not exemption from the effect.

Models are indistinguishable under unified tools but differ under native tools. A Kruskal–Wallis test finds no significant difference in unified scores ($H = 7.33$, $p = 0.40$), and 27 of the 28 pairwise Mann–Whitney comparisons are non-significant (the sole exception, GPT-5.4 vs. Gemini-3.7-flash at $p = 0.018$, does not survive Bonferroni correction). Under native tools the *same* test on the *same* sample strongly separates the models ($H = 90.75$, $p = 8.7 \times 10^{-17}$; 16 of 28 pairwise comparisons significant, means spreading 0.71–0.85). Tool-layer quality thus compresses the between-model gap: on clean, shared tools, between-model performance differences are substantially smaller for the

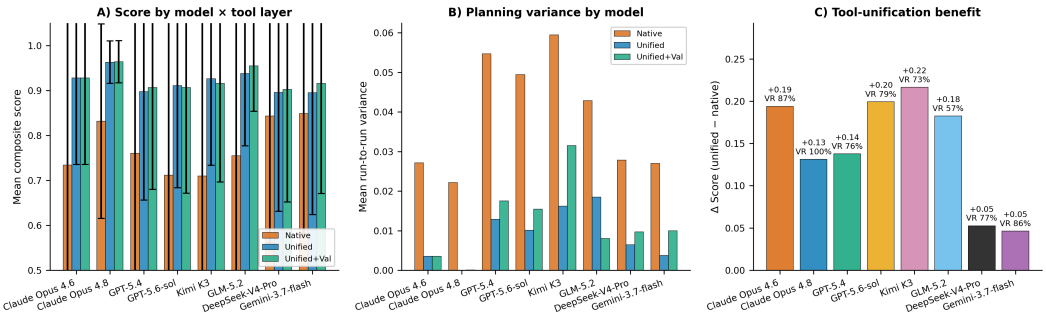

Figure 2: **Cross-model LLM planner comparison (Phase 3). A)** Mean composite score by model × tool layer; all models rise under unified tools and converge to a narrow band (0.89–0.96). **B)** Mean run-to-run variance by model; Claude Opus 4.8 reaches near-zero under unified tools, while the other models retain some residual variance concentrated on the hardest integration task. **C)** Tool-unification benefit (Δ score, unified − native) with variance-reduction (VR) annotations per model.

workflow-specified scRNA-seq tasks studied here, reducing not only inter-strategy variance (Phase 1) but also inter-model variance.

Complete variance elimination is model-dependent. Only Claude Opus 4.8 drives run-to-run variance to essentially zero under unified tools (100% reduction); the other seven retain measurable residual (57–87%), concentrated on the hardest integration task (`real_integ_02`, which withholds batch metadata): GPT-5.4, for example, still produces divergent pipelines there (unified inter-repeat variance = 0.216). Tool unification is therefore necessary but not always sufficient to eliminate variance; the remainder is a true planning-stability property that varies by model, invisible in the single-model experiment.

Against the rule-based optimum (0.970), Claude Opus 4.8 essentially closes the planning gap (unified 0.963, within 0.7%) while Gemini-3.7-flash, DeepSeek-V4-Pro, and GPT-5.4 retain the largest residuals (0.895–0.898; Appendix Table 13). The unified ordering does not match the native ordering, the two strongest models under native tools (Gemini-3.7-flash 0.849, DeepSeek-V4-Pro 0.843) are among the weakest under unified tools, so native scores conflate planning quality with error-interaction luck.

The per-category breakdown (Appendix Figure 10, Table 16) reveals model-specific signatures. *Claude Opus 4.6's native integration collapse* (0.26; only GPT-5.6-sol is comparable at 0.33, vs. 0.48–0.85 for the rest) traces to `ignore_batch_in_hvg` firing catastrophically on its stochastic multi-batch plans; under unified tools it recovers to 0.70 but remains its weakest category. *GPT-5.4 fails `real_core_03` on every repeat even under unified tools* (mean 0.00), consistently omitting multiple-testing correction, the clearest orchestration bottleneck surviving tool unification. *DeepSeek-V4-Pro and Gemini-3.7-flash are the only models whose integration category does not improve under unified tools* (0.55 → 0.49 and 0.59 → 0.50), driven by over-correction on `real_integ_04`. Interpretation scores converge to 0.90–0.98 across all eight models, and preprocessing/feature-engineering are likewise near-uniform. These categories have such well-specified workflows that, model-specific differences become small for these well-specified workflow categories.

Failure modes underlying Phases 1–3. Appendix Figure 12 maps validation errors across every task and tool layer, for both the rule-based harnesses and the eight LLM planners: errors concentrate in the native column and vanish entirely under unified and unified+validated tools. Quantitatively, across the 690 native rule-based runs the validators record 132 errors (one per affected run, 19.1% of runs, most commonly double normalization) plus 480 batch-effect warnings, consistent with the calibrated injection rates. The LLM planner shows a distinctive native failure pattern (up to seven errors per task): Claude-Opus-4.6's plans tend to include redundant normalization steps that compound with the double-normalization injection, but under unified tools the skill's already-normalized check makes the redundant planning harmless, tool quality compensating for planning imperfection. Integration tasks show the most dramatic native failures (LLM mean 0.194, effectively non-functional) via `ignore_batch_in_hvg`, recovering to 0.675 under unified tools, still weakest but now limited by planning rather than tool errors.

3.4 Tool Dominance Holds Only for Canonical Workflows

Phases 1–3 show that tool-layer reliability accounts for much of the observed variance on the standard scRNA-seq tasks studied here. But those tasks share a property: a *canonical* analytical workflow (QC → normalize → HVG → cluster → DE), we refer to these as workflow-specified (IID-like) tasks. This risks a circularity: studying only well-defined tasks may itself *manufacture* the "tools always win" result. Phase 4 tests the boundary by adding a four-task decision-structured (OOD-like) group to the 23-task core benchmark, in which the hard question is not *how* to execute a fixed pipeline but *which methodological decision* to make (e.g. "does this dataset even need batch integration?"). Decision-structured tasks carry an `expected_decision` ground truth and are scored on whether the plan makes the correct call; Workflow-specified tasks are scored by validator cleanliness as before. The resulting registry holds 27 tasks; because two of the original 23 are themselves decision-structured (the integration tasks asking whether batch correction is warranted), the split is 21 workflow-specified and 6 decision-structured. We run all three rule-based harnesses over this registry under native vs. unified tools ($3 \times 2 \times 27 \times 3 = 486$ runs).

The two task regimes behave differently (Figure 3). On IID tasks the Phase 1 result reproduces exactly: unifying tools collapses inter-harness variance from 0.0017 to zero, so even a naive fixed pipeline matches an adaptive planner. On OOD tasks, unification leaves variance essentially unchanged (0.1252 → 0.135); the residual concentrates in three decision tasks (`ood_de_requires_integration`, `ood_decide_integration_multi`, `real_integ_04`, each variance ≈ 0.25) where harnesses genuinely disagree about the right analytical decision. This is the **planning bottleneck in its pure form**: a variance source unified tools *cannot* touch because the difficulty is the decision, not the execution.

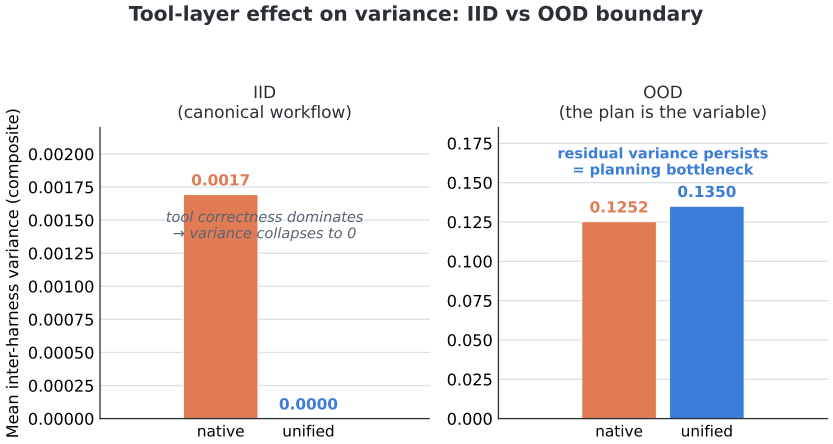

Figure 3: **The IID/OOD boundary (Phase 4).** Mean inter-harness variance under native vs. unified tools on the 21 IID (canonical-workflow) and 6 OOD (decision-structure) tasks. Unification collapses IID variance to zero but leaves OOD variance unchanged: on decision-structured tasks the residual is a planning bottleneck that tool quality cannot remove.

This upgrades the Phase 1 conclusion from a slogan ("tools dominate") into a *map*: tools dominate where a canonical workflow exists; strategy dominates where the task requires genuine methodological judgment. The OOD split thus shows exactly *when* tool-dominance holds and when it does not, answering the concern that our headline is an artifact of task selection.

3.5 Tool Bloat Is a Stability Tax on Registry-Querying Agents

A unified registry raises a practical question: how *many* tools should it contain? More tools mean broader coverage but also more opportunities to select the wrong one. Phase 5 measures this "tool bloat" effect and introduces an audit, the Minimal Sufficient Skill Set (MSSS), that recovers the smallest registry still covering the task distribution. We compare two registries, **minimal** (6 core skills) and **bloated** (13 skills: the 6 core plus 7 decoy/redundant tools), each run under two harnesses (≈ 540 runs). The harnesses differ in how they choose skills: `stochastic_selector` queries

the registry listing and samples among candidates (modeling an LLM/selection agent), whereas `adaptive_planner` hard-codes skill names and never consults the listing.

Tool bloat degrades stability, but *selectively*: the registry-querying `stochastic_selector`'s inter-run variance rises from 0.0773 to 0.0884, while the listing-blind `adaptive_planner` is entirely unaffected (0.0 in both) (Appendix Figure 4). The effect appears in *variance*, not mean score (decoys raise the *probability* of a wrong pick, not its certainty). Bloat is a stability tax paid only by agents that reason over the tool menu. The MSSS audit (`greedy_forward_select`) over the bloated 13-skill registry recovers exactly the 6 core skills at 100% task coverage and flags all 7 decoy/redundant tools as drop candidates. Registry parsimony is thus not a matter of taste but an auditable property with a measurable stability payoff. Figure 4 gives the full per-harness view of this bloat-induced degradation.

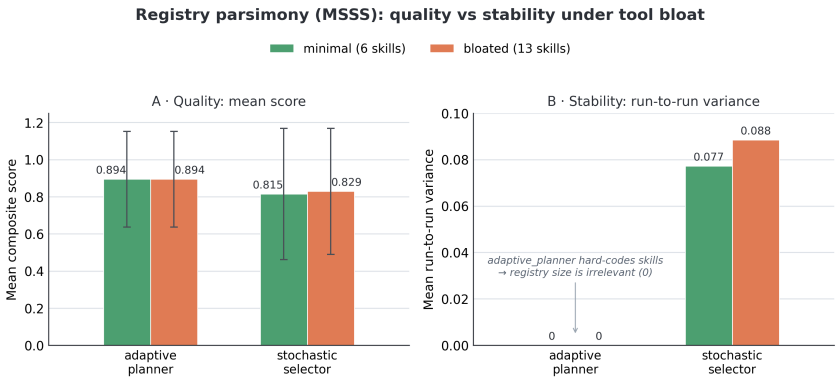

Figure 4: **Registry parsimony (Phase 5): the effect of tool bloat on quality and stability. A)** Mean composite score for the minimal (6-skill) vs bloated (13-skill) registry. **B)** Run-to-run variance for the same conditions. `adaptive_planner` hard-codes its skill sequence, so registry size leaves both its score and its variance unchanged; only `stochastic_selector`, which samples from the registry, degrades under bloat.

3.6 Strategy Wins Only Where an Oracle Exists

Phase 4 shows task structure (IID/OOD) is one boundary. Phase 6 isolates a second, orthogonal one: the presence of a runtime *oracle*, a correctness signal an agent can query *during* execution to self-correct (unit tests, exit codes, page state in general-agent domains (Zhou et al., 2024)). The general-agent literature concludes "strategy/feedback wins" in verifiable domains where such an oracle exists; sc-RNA-seq is a silent-failure domain where it typically does not. Rather than run SWE-bench, we manufacture the contrast *inside* biology: the **verifiable regime** is 4 clustering tasks carrying `expected_clusters` (a queryable target an `OracleGuidedHarness` sweeps Leiden (Traag et al., 2019) resolution to hit, API-free feedback-refinement à la Reflexion); the **silent regime** is 4 QC/normalization/HVG tasks with no target ($4 \times 8 \times 5 = 160$ runs).

The result is decisive (Figure 5). In the verifiable regime the oracle-guided harness leads the best non-oracle harness by **+0.57** (0.616 vs. 0.046; inter-harness variance 0.091): strategy matters enormously when a correctness signal is available. In the silent regime all four harnesses converge to 1.0 (variance 0), where strategy is drowned by tool correctness, exactly as in Phases 1 and 4-IID. The *same* `oracle_guided` strategy is dominant in one regime and irrelevant in the other, the only difference being oracle availability. This demonstrates the general-agent "strategy wins" rule is a function of *the oracle*, not of *the domain*: granting an oracle even inside a silent-failure domain revives the strategy advantage.

3.7 Dissect-Audit: A Reusable Diagnostic Protocol

The six phases yield not only findings but a reusable methodology, packaged as a one-command protocol (`dissect audit`) with three steps: (1) **tool-layer transplant**, which swaps an agent's native tools for the unified registry (via the MCP bridge) and measures the score uplift; (2) **preventive gating**, a `GatedRegistry` that refuses harmful operations *before* they run (two

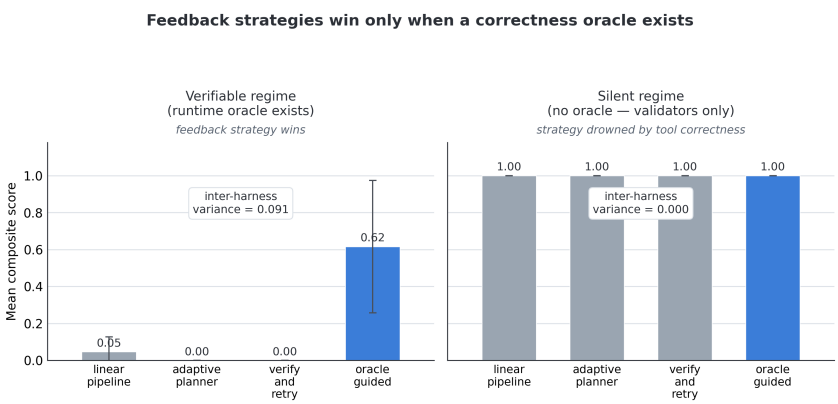

Figure 5: **The oracle boundary (Phase 6).** With a runtime oracle (verifiable regime), the oracle-guided harness leads all others by +0.57; without one (silent regime) every harness converges to 1.0. Strategy advantage is a function of oracle availability, not of the biological domain.

gates: `no_double_normalization`, `normalize_before_hvg`), the "mode 3" validators of Appendix F.2; and (3) the **MSSS audit** of §3.5. Reading the Phase 1 results through step (1) reveals a clean regularity.

Across the three rule-based harnesses the transplant uplift is uniformly positive (+0.100 to +0.127; Appendix Table 10), but three points cannot test whether uplift self-targets weaker agents, so we test that claim on the eight LLM planners of Phase 3, re-running the panel over the full 27-task set rather than the 23-task benchmark; its convergence result ($p = 0.61$ below) is thus consistent with §3.3's $p = 0.40$ at a different scope, not a competing estimate. The *variance* axis does not reach significance ($r = +0.46$, $p = 0.25$), but the *quality* axis does: corr(native mean score, uplift) $= -0.86$ ($p = 0.006$). The intervention is also uniformly beneficial: **every agent gains** (+0.08 to +0.20, mean +0.15; run-to-run variance reduced 42–100%, Wilcoxon $p = 0.008$). Unifying the tool layer then makes the eight models statistically indistinguishable (Kruskal–Wallis $H = 5.41$, $p = 0.61$, versus $H = 59.5$, $p = 1.9 \times 10^{-10}$ under native tools on the same panel). The recipe thus lifts every agent and lifts the weakest most, making it an actionable audit any bio-agent developer can run on their own system.

4 Conclusion

We introduce Dissect-Audit, to our knowledge the first controlled-experiment framework for causal study of performance variance in bio-agents. Across 8,550 runs on real-data sc-RNA-seq tasks (plus ~1,200 boundary-mapping runs), we hold the tool layer constant while varying planning strategy, and vice versa. In the regime characterizing most bio-analysis (in-distribution tasks, no runtime verification signal), tool-layer quality dominates: unifying tools drives inter-strategy variance to zero in our error-injection setup (Wilcoxon $p = 1.22 \times 10^{-4}$), a directional effect that reproduces across eight LLMs from six vendors (complete elimination being model-dependent), and validators add nothing once tools are reliable; only a 0.7% orchestration gap remains, concentrated in batch-correction judgment calls. This dominance is bounded. On out-of-distribution tasks hinging on methodological choice, unified tools cannot collapse variance (0.125 → 0.135), and with a runtime oracle, oracle-guided planning beats the best non-oracle strategy by +0.57: results are governed by oracle availability and IID/OOD, not bio-vs-general, reconciling our findings with the general-agent literature. Our packaged diagnostics (variance attribution, tool transplant, preventive gating, skill pruning) yield uniformly positive transplant uplift (+0.08–0.20) that self-targets the weakest agents (corr $= -0.86$, $p = 0.006$), arguing for shared validated tool registries as infrastructure and redirecting planning research toward OOD/oracle regimes. Concretely, our evidence delineates what tool unification alone can and cannot fix. *Solved by clean tools:* canonical-workflow tasks where inter-strategy variance falls to zero. *Not solved by clean tools:* tasks whose difficulty is a methodological *decision* and tasks requiring a runtime correctness signal, where planning strategy, not tool quality, remains the binding constraint.

AI USE STATEMENT

In this work, we used generative AI tools as a coding assistant during the development of the code framework, benchmark harnesses, and analysis scripts, and as a writing aid for polishing portions of the manuscript. We have not used generative AI tools to generate research ideas, hypotheses, or experimental designs, and generative AI was not used to collect or analyze experimental data; the large language models appearing in this paper (e.g., Claude, GPT, Kimi, GLM, DeepSeek, and Gemini models) are the *subjects* of the study, invoked through their public APIs as the planning components under test, not as assistants in producing this work. We have reviewed all AI-assisted work: LLM-assisted code was verified by running the released test suite and by reproducing the reported experiments end-to-end, and all AI-assisted text was read, edited, and checked against the underlying experiment logs by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Enis Afgan, Dannon Baker, Bérénice Batut, Marius Van Den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn Grüning, et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46 (W1):W537–W544, 2018.

Anthropic. Model context protocol specification. `https://modelcontextprotocol.io`, 2024.

Anthropic. Claude Science: An AI workbench for scientists, June 2026. URL `https://www.anthropic.com/news/claude-science-ai-workbench`. Accessed: 2026-09-03.

Reuben M Baron and David A Kenny. The moderator–mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology*, 51(6):1173–1182, 1986.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, volume 2025, pp. 96934–96990, 2025.

Alireza Ghafarollahi and Markus J. Buehler. SciAgents: Automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025. doi: 10.1002/adma.202413523.

Ali Essam Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J Szostkiewicz, Dmytro Shved, Gavin J Gyimesi, Jon M Laurent, Samantha M Wright, Muhammed T Razzak, et al. A multi-agent system for automating scientific discovery. *Nature*, 655(8122):497–505, 2026. doi: 10.1038/s41586-026-10652-y.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, et al. BLADE: Benchmarking language model agents for data-driven science. In *Findings of the association for computational linguistics: EMNLP 2024*, pp. 13936–13971, 2024.

Wenpin Hou and Zhicheng Ji. Assessing GPT-4 for cell type annotation in single-cell RNA-seq analysis. *Nature methods*, 21(8):1462–1465, 2024.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Ryan Li, Yusuf Roohani, Lin Qiu, Shiyi Cao, Gavin Li, Junze Zhang, Di Yin, Rick Wierenga, Deniz Kavi, Sherry Liu, Tianwei She, Shruti Marwaha, Jennefer N Carter, Xin Zhou, Matthew T Wheeler, Jonathan A Bernstein, Mengdi Wang, Peng He, Jingtian Zhou, Michael P Snyder, Le Cong, Aviv Regev, and Jure Leskovec. Autonomous biomedical research with an artificial intelligence agent. *Science*, 393: eadz4351, 2026. doi: 10.1126/science.adz4351. URL `https://www.science.org/doi/abs/10.1126/science.adz4351`.

Wolfgang Huber, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, et al. Orchestrating high-throughput genomic analysis with bioconductor. *Nature methods*, 12(2):115–121, 2015.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 754–762, 2014.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Hyun Min Kang, Meena Subramaniam, Sasha Targ, Michelle Nguyen, Lenka Maliskova, Elizabeth McCarthy, Eunice Wan, Simon Wong, Lauren Byrnes, Cristina M. Lanata, Rachel E. Gate, Sara Mostafavi, Alexander Marson, Noah Zaitlen, Lindsey A. Criswell, and Chun Jimmie Ye. Multiplexed droplet single-cell RNA-sequencing using natural genetic variation. *Nature Biotechnology*, 36(1): 89–94, 2018. doi: 10.1038/nbt.4042.

Ilya Korsunsky, Nathan Millard, Jean Fan, Kamil Slowikowski, Fan Zhang, Karl Wei, Yuri Baglaenko, Michael Brenner, Po-ru Loh, and Soumya Raychaudhuri. Fast, sensitive and accurate integration of single-cell data with Harmony. *Nature Methods*, 16(12):1289–1296, 2019.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, 2023.

Orion Li, Vinayak Agarwal, Summer Zhou, Ashwin Gopinath, and Timothy Kassis. K-Dense Analyst: Towards fully automated scientific analysis. *arXiv preprint arXiv:2508.07043*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Malte D Luecken and Fabian J Theis. Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular systems biology*, 15(6):MSB188746, 2019. doi: 10.15252/msb.20188746.

Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B Hall, Peter C van Dyken, Christopher H Tomkins-Tinch, Vanessa Sochat, Jan Forster, Filipe G Vieira, Christian Meesters, et al. Sustainable data analysis with Snakemake. *F1000Research*, 10:33, 2021.

Douglas C Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 10th edition, 2020.

Odhran O'Donoghue, Aleksandar Shtedritski, John Ginger, Ralph Abboud, Ali Ghareeb, and Samuel Rodriques. BioPlanner: Automatic evaluation of LLMs on protocol planning in biology. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2676–2694, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.162.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4020.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Yuanhao Qu, Yingzhou Lu, Xinming Tu, Serena Zhang, Tianwei She, Alexander Glenn Shaw, Jou-Ho Shih, Bingqing Zhao, Minjie Shen, Haochen Yang, Jielin Yan, Rongchuan Zhang, Xinze Wu, Tingting Li, Bin Zhou, Ning Wang, Adam Ma, Le Cong, Xiaobo Hu, Yuan Jiang, Jiayun Dong, Tao Peng, Jure Leskovec, and Kexin Huang. BiomniBench: Process-level evaluation of LLM agents for real-world biomedical research. *bioRxiv*, 2026. doi: 10.64898/2026.05.12.724604.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023. doi: 10.52202/075280-2997.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023.

Synthetic Sciences. OpenScience: The open-source AI workbench for scientific research, 2026. URL `https://openscience.sh/`. Accessed: 2026-09-03.

Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):5233, 2019.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

F Alexander Wolf, Philipp Angerer, and Fabian J Theis. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, 2018.

Yuxuan Wu and Fuchou Tang. scExtract: leveraging large language models for fully automated single-cell RNA-seq data annotation and prior-informed multi-dataset integration. *Genome Biology*, 26(1):174, 2025. doi: 10.1186/s13059-025-03639-x.

Yihang Xiao, Jinyi Liu, Yan Zheng, Shaoqing Jiao, Jianye Hao, Xiaohan Xie, Mingzhi Li, Ruitao Wang, Fei Ni, Yuxiao Li, Zhen Wang, Xuequn Shang, Zhijie Bao, Changxiao Yang, and Jiajie Peng. CellAgent: LLM-driven multi-agent framework for natural language-based single-cell analysis. In *Proceedings of the 14th International Conference on Learning Representations*, 2026.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations*, 2023b.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

A Summary of Key Findings

To the best of our knowledge, Dissect-Audit provides the first causal evidence decomposing performance variance in bio-agents into tool-layer and strategy-layer contributions, and several findings emerge.

1. **Tool unification eliminates inter-strategy variance, and validators add nothing on clean tools** (Phases 1–2): unifying the tool layer drives inter-harness variance to zero (Wilcoxon $p = 1.22 \times 10^{-4}$, 100% reduction), and adding output validation on top yields numerically identical outcomes, so validators are reactive safeguards rather than proactive quality enhancers.
2. **The pattern generalizes across eight frontier LLMs and compresses the between-model gap** (Phase 3): all eight models improve significantly under unified tools (Δ +0.05 to +0.22, Cohen's $d = 0.17$–0.84, $p < 10^{-4}$ in every case), and models that differ markedly under native tools become statistically indistinguishable once tools are unified (Kruskal–Wallis $p = 0.40$).
3. **The residual orchestration gap is small, real, and model-specific** (Phase 3): only Claude Opus 4.8 reaches exactly zero run-to-run variance (100% reduction) while the other seven models (including GPT-5.4 at 76% and Kimi K3 at 73%) retain residual planning noise, and unified scores span 0.895–0.963 against the rule-based optimum of 0.970, with GPT-5.4's shortfall tracing to intermittently omitting multiple-testing correction.
4. **Tool dominance is bounded by task structure** (Phase 4): unified tools collapse inter-harness variance on IID canonical-workflow tasks (0.0017 → 0.0000) but leave it essentially unchanged on OOD decision tasks (0.1252 → 0.135), the pure planning bottleneck that tool correctness cannot remove.
5. **Registry parsimony is auditable and matters** (Phase 5): tool bloat (13 vs. 6 skills) raises inter-run variance for registry-querying agents (0.0773 → 0.0884) but not for listing-blind ones, and the MSSS audit recovers the minimal sufficient 6-skill set at 100% coverage.
6. **The decisive boundary is the oracle, not the domain** (Phase 6): a runtime oracle inside sc-RNA-seq revives the strategy advantage (oracle-guided +0.57 over the best non-oracle harness), while without one all strategies converge to 1.0.
7. **Transplant uplift is uniformly positive and self-targeting** (Dissect-Audit): every one of the eight LLM planners gains from a shared validated registry (+0.08 to +0.20, mean +0.15; run-to-run variance reduced 42–100%, Wilcoxon $p = 0.008$), and the gain is largest for the weakest agents (corr(native mean score, uplift) $= -0.86$, $p = 0.006$). The three rule-based harnesses are too few to test this (N=3): the *variance* axis does not reach significance across the eight models ($r = +0.46$, $p = 0.25$), and the self-targeting effect lives on the quality axis.

B Experimental Setup and Protocols

B.1 Benchmark Task Suite

Tasks are constructed to satisfy three properties required for a valid causal experiment: *known failure-mode association* (each targets documented methodological errors, so native injection produces task-relevant failures with ecological validity), *category coverage* (the full sc-RNA-seq pipeline, so findings are not an artifact of one operation type), and a *difficulty gradient* (single-operation easy tasks through multi-step expert tasks where errors compound). The 23 real-data core tasks are drawn from a 5-category taxonomy aligned with standard sc-RNA-seq practice; a further 4-task *OOD decision group* (where the correct plan, not clean execution, is the hard part) is held out for the boundary experiments (Phases 4–6) and the Dissect-Audit, giving 27 tasks in total (Table 1). Two of the 23 core tasks are themselves decision-structured, so across all 27 tasks the regime split is 21 IID and 6 OOD.

Each task is a `BenchmarkTask` specifying identifier, description, category, difficulty, dataset path, required skills (expected correct pipeline), known failure modes, optional task-specific configuration, and optional ground truth. Two of the 23 tasks carry external ground truth (`real_core_01` clustering; `real_interp_03` markers), enabling absolute quality measurement; the remaining 21 are scored through the composite validator (§C.7). This is deliberate: ground truth for most sc-RNA-seq tasks is unavailable or contested (no universally "correct" clustering), making process-based evaluation (did the agent avoid documented errors?) more appropriate than output-based. Representative tasks spanning the difficulty range are described below (§B.2).

Table 1: Complete benchmark task suite: the 23-task core benchmark (Phases 1–3; 21 canonical-workflow + 2 decision-structured tasks) plus a 4-task OOD decision group (task-structure OOD; held out for Phases 4–6 and DISSECT-AUDIT), 27 tasks total. The two decision-structured core tasks are the integration tasks asking whether batch correction is warranted, so the regime split across all 27 tasks is 21 IID and 6 OOD.

Category	*n*	**Difficulty Range**	**Key Failure Modes Tested**
Core benchmark (23 tasks; 21 canonical-workflow + 2 decision-structured)			
Preprocessing	6	easy–hard	Wrong MT prefix, over-filtering, double normalization, skip-normalize detection
Feature Engineering	4	easy–hard	Batch-unaware HVG, overly stringent filtering, integration-prep feature selection
Core Analysis	5	easy–hard	Missing multiple testing correction, multi-batch DE without proper modeling
Integration	4	medium–expert	Unhandled batch effects, missing batch info detection, over-correction
Interpretation	4	medium–expert	Hallucinated DOIs, marker gene validation, scalability on large datasets
OOD decision group — correct plan *is non-obvious; scored on the decision, not execution (4 tasks)*			
Integration	2	hard	Integrate-or-not: single-batch data (must *not* integrate) vs. multi-donor data (must integrate)
Preprocessing	1	hard	Re-normalize already-processed data (must *not*; double-normalization)
Core Analysis	1	expert	Integrate before DE on multi-batch data (else batch-confounded markers)

B.2 REPRESENTATIVE TASKS

Each `BenchmarkTask` specifies an identifier, description, category, difficulty, dataset path, required skills (the expected-correct pipeline), known failure modes, optional task-specific configuration, and optional ground truth. Representative tasks across the difficulty range:

- `real_preproc_01` (easy, PBMC3k): standard QC + normalization; tests the wrong-mito-prefix injection.
- `real_preproc_04` (hard, PBMC3k annotated): the data arrives already processed; the agent must detect this and *skip* re-normalization, one of the two highest-variance tasks (native variance 0.0085).
- `real_feat_02` (medium, Kang2018): batch-aware HVG selection; tests `ignore_batch_in_hvg` on genuine multi-batch data.
- `real_core_01` (easy, PBMC3k annotated): Leiden clustering scored by cluster-count ratio against annotated ground truth, one of two ground-truth tasks.
- `real_integ_04` (expert, Kang2018): integration where over-correction is the failure mode; an OOD decision task.
- `real_interp_03` (medium): marker-gene identification scored by recall against curated marker lists, the second ground-truth task.

Two of the 23 tasks carry external ground truth (`real_core_01`, `real_interp_03`); the remaining 21 are scored through the composite validator. This is deliberate: ground truth for most sc-RNA-seq tasks is unavailable or contested (no universally "correct" clustering), making process-based evaluation (did the agent avoid documented errors?) more appropriate than output comparison.

B.3 DATASETS

Four real sc-RNA-seq datasets provide the data substrate (Table 2). **PBMC3k** (10x Genomics) is the canonical single-batch tutorial dataset, the baseline for preprocessing, feature-engineering, and core-analysis tasks. **Kang2018** (GSE96583) provides genuine multi-batch structure (stimulated vs. control PBMCs), triggering the `ignore_batch_in_hvg` injection (rate 0.8) with maximal effect.

PBMC3k annotated arrives already processed, testing whether the agent recognizes processed data and avoids re-normalization. **PBMC68k reduced** (downsampled to 700×765, pre-scaled with negative values) tests both scalability reasoning and data-state detection (the normalization validator must accept z-scored data as legitimate).

Table 2: Datasets used in the benchmark. The Tasks column sums to the 23-task core benchmark.

Dataset	Cells	Genes	Batches	Source	Tasks
PBMC3k (raw)	2,700	3,000	1	10x Genomics	12
Kang2018 (multi-batch)	5,000	17,194	2	GSE96583	8
PBMC3k (annotated)	2,621	654	1	10x (processed)	1
PBMC68k (reduced)	700	765	1	10x (subsampled)	2

B.4 Experimental Design Matrix

The study comprises six phases in two groups. The *core factorial experiment* (Phases 1–3) isolates the tool-layer effect: Phase 1 is the primary causal-diagnosis experiment, Phase 2 measures the marginal impact of the validation layer, and Phase 3 replicates the core comparison with real LLM planners across eight models. A second group of *boundary experiments* (Phases 4–6) then maps *when* tools versus strategy dominate: the IID/OOD task boundary (Phase 4), tool-registry parsimony (Phase 5), and the verifiable/oracle regime (Phase 6). Because each boundary experiment is best understood alongside its result, we introduce its design together with its findings in Section 3; the Combined Design summary (Table 3) accounts for all six.

All three core phases treat the tool layer as the manipulated factor while holding the rest of the configuration fixed; planning strategy and tool layer are within-subjects, task identity is a blocking variable controlling for difficulty, and the 10 repeats supply statistical power. **Phase 1** (primary) crosses the three rule-based strategies with the two tool layers (native, unified) over the 23 tasks. Its outcome is the per-task inter-harness variance of composite score, and we test H_0 : Var(native) = Var(unified) against the one-sided H_1 : Var(native) > Var(unified). **Phase 2** contrasts unified with unified+validated on the same design, testing whether validation improves the mean score over the unified baseline (one-sided). **Phase 3** fixes the harness to the LLM planner and crosses eight frontier models with the three tool layers (810 runs per model over the full 27-task panel, 6,480 total; the scope-matched 23-benchmark subset analyzed as the Phase-1 replication is 690 per model, 5,520 total, with the 4 OOD decision tasks held out to §3.7). Two choices make Phase 3 a genuine generalization test: non-zero temperature ensures repeats produce genuinely different plans (so variance reduction is not a deterministic-execution artifact), and running eight models separates a property of the shared skill/validator layer from any single model's idiosyncrasy.

B.4.1 Combined Design

In practice the core factorial study executes as two independent sessions—the rule-based experiment (Phases 1 and 2) and the LLM experiment (Phase 3, eight models)—for 8,550 runs. The boundary experiments (Phases 4–6) add a further 1,186 runs, bringing the study to 9,736 runs in total. Table 3 summarizes both blocks.

The study is deliberately cheap to replicate. The rule-based Phases 1–2 and all three boundary experiments run CPU-only ($0 API cost); the entire LLM API bill is under $70, confined to Phase 3 (~$8 per model). The ~51h grand total is a sequential wall-clock upper bound dominated by API latency; running the eight Phase-3 models in parallel collapses it to well under half a day. Low replication cost is an explicit design goal for a framework intended for community use.

B.5 Error Injection, Statistics, and Reproducibility

The failure *types* we inject are documented in agent evaluations (Qu et al., 2026) (Table 4); the specific injection *rates* are our calibration choice, set deliberately high (0.4–0.8) so that most tasks experience at least one error across the 3 harnesses and 10 repeats and to give adequate statistical power (lower pilot rates of 0.2–0.6 produced zero variance on 13/23 tasks). The magnitude of the effect is therefore

Table 3: Combined experimental design and cost summary. The core factorial experiment (Phases 1–3) isolates the tool-layer effect; the boundary experiments (Phases 4–6) map its limits and substitute their governing factor for the tool-layer column—registry size in Phase 5, and regime/oracle availability in Phase 6. The final column reports wall-clock time and LLM API cost; the Phase 3 figure is the sum over all eight models run sequentially and is dominated by API latency (2.3–9.8h per model), so it shrinks substantially when models are run in parallel. †Phase 3 executes the full 27-task panel (6,480 runs); its main analysis (§3.3) is limited to the 23 benchmark tasks (5,520 runs) to match the Phase-1 replication scope, with the 4 OOD decision tasks held out and analyzed in §3.7. Phase 4's "6 OOD" is a split of 4 structural-OOD decision tasks plus 2 borrowed integration tasks (`real_integ_02`, `real_integ_04`); it is a subset of the 27-task panel and not counted separately. Phase 6's 8 tasks are 4 verifiable + 4 silent.

Phase	Harnesses	Tool Layers	Tasks	Repeat	Runs	Compute (API cost)
Core factorial experiment						
Phase 1+2	3 rule-based	3 (native, unified, unified+validated)	23	10	2,070	~2.3h CPU ($0)
Phase 3 (LLM)	1 (LLM Planner × 8 models)	3 (native, unified, unified+validated)	27†	10	6,480	~45.6h wall-clock, 8 models ($64)
Subtotal					**8,550**	**~47.9h (~$64)**
Boundary experiments						
Phase 4 (IID/OOD)	3 rule-based	2 (native, unified)	27 (21+6)	3	486	~0.9h CPU ($0)
Phase 5 (parsimony)	2	2 (6- vs. 13-skill registry)	27	5	540	~1.5h CPU ($0)
Phase 6 (oracle)	4	1 (unified)	8 (4+4)	5	160	~0.3h CPU ($0)
Subtotal					**1,186**	**~2.7h ($0)**
Total					**9,736**	**~50.6h (<$70)**

calibration-dependent (§G), while its direction is not. Each repeat uses seed = 42 + repeat_index, giving within-repeat determinism (exact replication) and across-repeat variance; the native RNG is re-seeded per run to prevent state leakage.

Table 4: Documented failure types instantiated by the native error-injection layer. Injection rates are our calibration choice (see §B.5), not measured frequencies.

Error Type	Rate	Documented failure type / rationale
Double normalization	0.70	Documented normalization failure; agents are known to mishandle normalization (Qu et al., 2026)
Batch-unaware HVG	0.80	Ignoring batch effects is a documented agent failure (Qu et al., 2026); especially damaging on multi-batch data
Missing multiple testing	0.60	Multiple-testing correction is an evaluated methodological requirement (Qu et al., 2026); commonly omitted in automated DE
Wrong mito prefix	0.40	Organism-specific prefix (`MT-` vs `mt-`) in the standard QC workflow (Wolf et al., 2018); wrong prefix silently voids mitochondrial QC on human data
Over-filtering	0.40	Overly stringent QC thresholds remove rare cell populations (Luecken & Theis, 2019); harder to detect

Statistical analysis. Our pre-registered primary test for Phase 1 is the one-sided Wilcoxon signed-rank test on paired inter-harness variance differences (native − unified), paired by `task_id`; we use it because the differences are zero-inflated (10/23 tasks have no native variance) and right-skewed,

making it more robust than the paired t-test (reported as a secondary check). We quantify magnitude with the variance-reduction ratio $1 - \overline{\text{Var}}_{\text{unified}}/\overline{\text{Var}}_{\text{native}}$ and, for Phase 3, Cohen's d. Across the three pre-specified primary comparisons (Phases 1–3) we apply a Bonferroni correction ($\alpha_{\text{adj}} = 0.017$); all reported effects survive it. The 10-repeat design was fixed *a priori* from a 3-repeat pilot ($p = 0.059$) whose effect size indicated ∼10 repeats would give decisive power.

C The Dissect-Audit Framework

C.1 Design Principles

Dissect-Audit is organized around a single design objective: to enable controlled causal experiments on agent architectures by separating the independent variables (tool layer, planning strategy) from confounds (data complexity, implementation quality). Three architectural principles serve this objective. The first is *vertical composability*: the system is a strict stack (skills at the bottom, validators in the middle, harnesses at the top) with clean interfaces between layers, so that a harness operates identically regardless of which tool layer sits beneath it, and validators observe the analysis context without modifying execution flow (unless the harness explicitly queries them, as in verify-and-retry mode). The second is an *identical execution substrate*: every planning strategy, whether a fixed-order pipeline, an adaptive planner, or a Claude-Opus-4.6 agent, executes its planned steps through the same `SkillRegistry.invoke()` entry point, which guarantees that observed differences in output quality originate from planning decisions rather than from implementation divergence in the execution engine. The third is *calibrated error injection*: the native tool layer does not introduce random noise but injects specific, documented failure modes (double normalization, batch ignoring, wrong mitochondrial prefix) at calibrated rates derived from empirical agent failure analyses. This last principle makes the native-versus-unified comparison causally interpretable, with native representing what agents observably do wrong and unified representing what standardization prevents.

C.2 Architecture Overview

Figure 6 illustrates the four-layer architecture. The independent variables of the experiment are annotated: IV_1 is the planning strategy (which harness), IV_2 is the presence/absence of output validation, and IV_3 is the tool layer condition (native with error injection vs. unified clean skills). The dependent variable is the composite quality score of the analysis output, measured through validation findings and (where available) ground-truth comparison.

C.3 Skill Registry

Each skill inherits from an abstract `Skill` base class exposing typed metadata, a `validate_input` pre-execution hook, an `execute` method (typically wrapping a Scanpy function), and a `validate_output` post-execution hook; `run()` composes these as `validate_input` → `execute` → `validate_output`, and harnesses always call `run()`. The proof-of-concept library implements 6 skills covering the standard sc-RNA-seq workflow (Table 5). Each skill is deterministic under identical input, so under the unified tool layer any observed variance must originate from the planning strategy—which skills are called, in what order, with what parameters—not from execution stochasticity. The `SkillRegistry` is the single entry point for all tool execution across all harnesses and conditions, and additionally exports OpenAI function-calling schemas for LLM consumption (§C.8 details its API).

Table 5: Implemented sc-RNA-seq skills in the Dissect-Audit registry.

Skill	Category	Operation	Key Parameters
`scrna_qc_metrics`	Quality Control	QC metrics + filtering	`mito_prefix`, `min_genes`
`scrna_normalize`	Normalization	Library-size norm + log1p	`target_sum`
`scrna_hvg`	Feature Selection	Highly variable genes	`n_top_genes`, `batch_key`
`scrna_cluster`	Clustering	Leiden community detection	`resolution`
`scrna_de`	Differential Expression	Wilcoxon + correction	`method`, `corr_method`
`scrna_batch_integration`	Integration	Harmony batch correction (Korsunsky et al., 2019)	`batch_key`

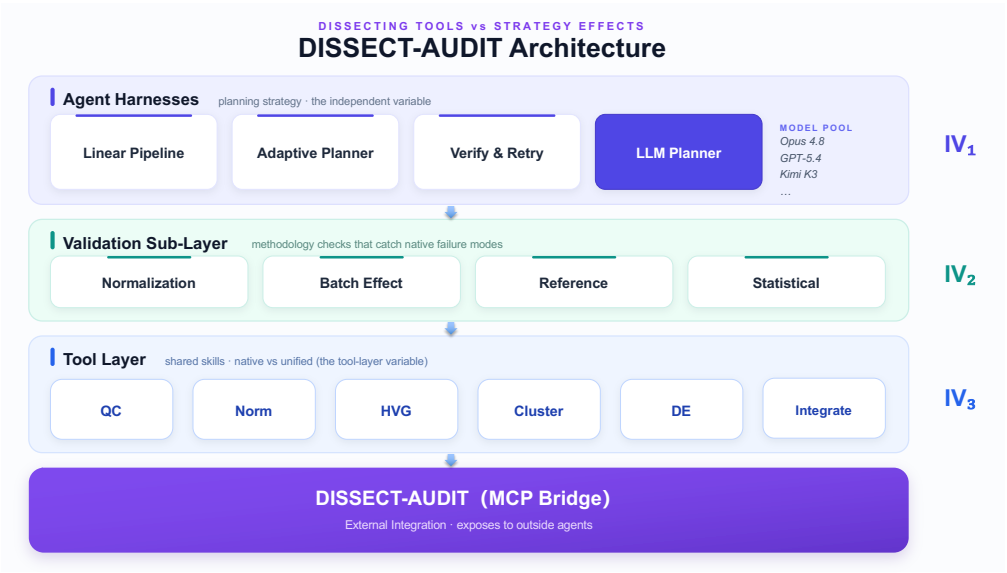

Figure 6: **DISSECT-AUDIT framework architecture.** Four layers compose vertically: agent harnesses (IV_1), validation sub-layer (IV_2), tool layer (IV_3), and MCP bridge for external integration. Each independent variable can be manipulated independently.

C.4 TOOL LAYER MODES

The tool layer is the primary independent variable in Phase 1, with three conditions. **Native (error-injecting)** mode wraps each clean operation but, with calibrated probability, injects a documented agent failure mode (Table 6). Injection is governed by a seeded `numpy.random.RandomState`, ensuring per-run reproducibility while generating inter-run variance through seed variation; different harnesses executing the same task encounter different error patterns (the seed advances differently along different execution paths), which is the inter-harness variance our experiment detects. **Unified (clean)** mode provides best-practice implementations (checking for already-normalized data before re-normalizing, respecting `batch_key` in HVG selection, always applying Benjamini–Hochberg correction, and using the correct mitochondrial prefix), so the tool layer is constant across harnesses and only the planned skills and parameters differ. **Unified + Validated** adds active output validation (§C.5) after execution; for verify-and-retry this enables intermediate error recovery, while for non-adaptive harnesses findings are recorded but do not alter execution.

Table 6: Injected error types in Native mode.

Error Type	Rate	Mechanism	Downstream Effect
Double normalization	0.70	Applies `normalize_total` to already-normalized data	Expression values in 15–100 range (should be < 15)
Batch-unaware HVG	0.80	Ignores `batch_key` parameter in multi-batch data	HVG driven by batch variation, not biology
Missing multiple testing	0.60	Reports raw p-values without Benjamini–Hochberg	Inflated false discovery rate
Wrong mito prefix	0.40	Uses `mt-` instead of `MT-` for human data	QC metrics miss mitochondrial genes entirely
Over-filtering	0.40	Applies overly stringent cell/gene thresholds	Removes rare cell types (DC, platelets)

C.5 OUTPUT VALIDATORS

The validation sub-layer encodes domain-specific methodological knowledge as automated checks that detect specific failure patterns documented in agent evaluations (Qu et al., 2026), rather than

generic assertions. Four validators cover the principal failure modes: a *normalization* validator (double normalization, caught via both a step-count and a value-range signature; double log-transform; negative values in count data, with a data-state detector that excludes legitimately z-scored inputs); a *batch-effect* validator (unhandled multi-batch data and non-batch-aware HVG, with batch columns detected heuristically from `obs`); a *reference* validator (DOI-format and method-citation checks against a curated method→reference map); and a *statistical* validator (missing multiple-testing correction, excessive significance). The `CompositeValidator` aggregates their findings by three severity levels: *error* (invalidating, e.g. double normalization), *warning* (likely but not fatal, e.g. missing batch correction), and *info*. The composite score (§C.7) combines these counts into a continuous quality measure; per-validator internals are in §C.9.

C.6 Agent Harnesses

Each harness implements a distinct planning strategy while sharing the execution substrate; the abstraction requires only `plan(task)` → `list[step]`, with a default `execute_task()` handling skill invocation, AnnData propagation, and validation. We study four. The **Linear Pipeline** executes a fixed order (QC → Normalize → HVG → Cluster → DE) regardless of data, modeling naive "textbook" agents and serving as the robustness baseline. The **Adaptive Planner** inspects task metadata before constructing the pipeline (inserting batch integration when a `batch_key` is present, stopping early for `qc_only` tasks), modeling data-aware agents like Biomni. The **Verify-and-Retry** harness uses the adaptive plan but validates after each step and retries (up to 2 attempts) with heuristically adjusted parameters when errors are detected (e.g. double normalization → `skip=True`), modeling Reflexion-style self-correction. Finally, the **LLM Planner** sends the task description and tool definitions (in OpenAI function-calling format) to a Claude-Opus-4.6 agent, which returns an ordered plan as JSON; execution then proceeds through the same `execute_task()` as all other harnesses, so only the planning source differs. Temperature is fixed at 0.7, an intentional choice: non-zero temperature generates run-to-run plan variance, and we measure whether tool-layer unification reduces the *outcome* variance this planning variance induces. Failed LLM calls yield an empty plan (score 0), recorded as a legitimate agent-failure data point.

Under native tools, verify-and-retry partially recovers from injected errors, scoring higher than the non-adaptive harnesses (0.870 vs. 0.843); under unified tools there are no errors to recover from and all harnesses converge (0.970). This differential is precisely the inter-harness variance that tool unification eliminates. Prompt design and full harness pseudocode are in §C.10.

C.7 Scoring Mechanism

AnnData propagates through the step pipeline via a shared `context["adata"]` key, so errors compound: a normalization error in step 2 corrupts all downstream operations. This compounding is realistic (early errors propagate silently in real deployments) and is what makes the native-vs-unified difference measurable through downstream quality rather than only at the injection point. The composite score integrates validation findings into a continuous quality measure:

$$\text{score}_{\text{composite}} = \max\big(0,\ \mathbb{1}[\text{no execution error}] \ - \ 0.2 \cdot n_{\text{errors}} \ - \ 0.05 \cdot n_{\text{warnings}}\big) \tag{1}$$

where $\mathbb{1}[\cdot]$ is 0 if the pipeline crashed and 1 otherwise. It prioritizes methodological correctness (no errors) over optimality (no warnings): catastrophic failures such as double normalization matter more than suboptimal choices such as skipping batch-aware HVG. For the two tasks with external ground truth the score instead reflects reference agreement: `real_core_01` (clustering) by cluster-count ratio against `pbmc3k_annotated.h5ad` labels, and `real_interp_03` (markers) by recall against curated marker lists. To serve external agents beyond the four built-in harnesses, Dissect-Audit also exposes the registry through the Model Context Protocol (MCP) via an `MCPToolBridge` and stdio server (§C.11).

C.8 SkillRegistry API

Every skill subclasses an abstract `Skill` exposing (i) typed metadata (`name`, `category`, input/output schemas), (ii) a `validate_input(context)` pre-execution hook, (iii) an `execute(context)` method wrapping a Scanpy operation, and (iv) a

`validate_output(context)` post-execution hook. The public entry point `run()` composes these as `validate_input` → `execute` → `validate_output`; harnesses always call `run()`, never `execute()` directly, so validation is never bypassed. The `SkillRegistry` is the single lookup and execution point across all harnesses and tool-layer conditions: `registry.invoke(name, **params)`. It additionally exports OpenAI function-calling schemas via `registry.to_openai_tools()`, which is what the LLM planner consumes. AnnData objects propagate between steps through a shared `context["adata"]` key and are stripped before JSON serialization, so no AnnData ever enters the result files.

C.9 Validator Internals

The four validators of §C.5 implement the following checks.

- **Normalization validator.** Flags double normalization via two independent signatures: a *step-count* signature (two `normalize_total` calls in the trace) and a *value-range* signature (post-normalization expression maxima in the 15–100 range, above the <15 expected after a single log1p). Also flags double log-transform and negative values in putative count data. A data-state detector excludes legitimately z-scored inputs (e.g. PBMC68k reduced) from the negative-value check so pre-scaled data is not falsely flagged.
- **Batch-effect validator.** Detects unhandled multi-batch data and non-batch-aware HVG selection. Batch columns are found heuristically from `obs` (columns whose name matches a batch vocabulary and whose cardinality is 2–50). Emits a *warning* (not an error) when batch structure is present but no integration/batch-aware HVG step ran.
- **Reference validator.** Checks generated citations for DOI-format validity and verifies method→reference consistency against a curated map, catching hallucinated DOIs and mismatched method citations in interpretation tasks.
- **Statistical validator.** Flags missing multiple-testing correction (raw p-values reported without Benjamini–Hochberg) and implausibly high significant-gene counts.

The `CompositeValidator` runs all four and aggregates findings by severity: *error* (invalidating, e.g. double normalization), *warning* (likely-but-not-fatal, e.g. missing batch correction), and *info*. These counts feed the composite score of Eq. 1.

C.10 Harness Pseudocode and LLM Prompt

All harnesses share a default `execute_task()` that iterates the planned steps, calls `registry.invoke()` per step, propagates AnnData, and collects validation findings; only `plan(task)` → `list[step]` differs.

- **Linear Pipeline.** Returns a fixed step list (QC → Normalize → HVG → Cluster → DE) regardless of task metadata.
- **Adaptive Planner.** Inspects task metadata before building the plan: inserts `scrna_batch_integration` when `batch_key` is present; stops after QC for `qc_only` tasks; branches on the presence of `label_key`.
- **Verify-and-Retry.** Uses the adaptive plan but, after each step, runs the `CompositeValidator`; on an error it retries (up to 2 attempts) with heuristically adjusted parameters (e.g. double normalization detected → re-invoke with `skip=True`).
- **LLM Planner.** Sends the task description plus `registry.to_openai_tools()` to the model, which returns an ordered plan as JSON; execution then proceeds through the same `execute_task()`. Temperature is fixed at 0.7 to generate run-to-run plan variance. A failed or empty LLM response is recorded as an agent-failure data point (score 0). Models that reject `temperature=0.7` are retried once without the parameter.

C.11 MCP Bridge

To serve external agents beyond the four built-in harnesses, an `MCPToolBridge` exposes the skill registry as Model Context Protocol tool definitions and runs a stdio server. Each skill's input schema becomes an MCP tool schema, so any MCP-capable agent can call the unified, validated registry regardless of its LLM backbone or planning architecture, the technical template for the shared-registry proposal of §F.1.

D Supplementary Tables

This appendix collects the per-harness, per-task, per-category, and per-model breakdowns referenced from §3.

Positioning of this work. Table 7 situates Dissect-Audit relative to existing approaches along three dimensions. Dissect-Audit is unique in providing causal isolation of both tool-layer and strategy-layer effects through controlled factorial intervention. While narrower in domain (sc-RNA-seq only), the methodological template generalizes, and any domain with standardizable tools and multiple plausible strategies can apply the same experimental design.

Table 7: Comparison with related evaluation frameworks.

Framework	Domain	Evaluation Mode	Can Isolate Tool Effect?	Can Isolate Strategy Effect?
BiomniBench	Biology (multi-modal)	Observational	No	No
ScienceAgentBench	Science (data-driven)	Observational	No (correlational)	No
BioPlanner	Biology (protocols)	Observational	No	Partially (plans only)
AgentBench	General (8 environments)	Observational	No	Partially (varies LLM)
SWE-bench	Software engineering	Observational	Fixed (standardized)	Partially
ToolBench	Tool use	Ablation	Tool necessity only	No
Dissect-Audit	**sc-RNA-seq**	**Factorial experiment**	**Yes (causal)**	**Yes (causal)**

Table 8: Per-harness performance under the native (error-injecting) tool layer (Phase 1).

Harness	Mean Score	Std	Pass Rate (%)	Strategy
Linear Pipeline	0.843	0.249	73.5	Fixed order, no adaptation
Adaptive Planner	0.843	0.249	73.5	Data-aware, adapts ordering
Verify & Retry	0.870	0.278	95.7	Validates + retries on error

Table 9: Highest-variance tasks under native tools (Phase 1). Native per-task variance takes only three non-zero values across the 23 tasks; the remaining ten tasks are exactly zero.

Task ID	Native Var.	Category / Difficulty	Why
`real_preproc_04`	0.0085	Preprocessing / Hard	Detect already-processed data; verify-and-retry collapses to 0.0 vs. 0.16 for the rule-based plans
`real_interp_04`	0.0085	Interpretation / Hard	PBMC68k scale test; error compounding across many steps
six tied tasks	0.0065	Core / Interp.	`real_core_01/02/03/05`, `real_interp_01/03`: verify-and-retry recovers to 1.0 vs. 0.86

E Supplementary Figures

The following figures support the analyses above.

F Discussion

F.1 Implications for Agent Design

Our results argue for a specific priority ordering in bio-agent development. The first priority is to standardize the tool layer: before investing in planning sophistication, ensure that the underlying analysis operations are correct, validated, and consistent. Our data shows that a naive fixed-order pipeline on clean tools (score = 0.970) outperforms a sophisticated self-correcting agent on unreliable tools (score = 0.870). Tool quality is the floor, and strategy quality determines how far above that floor an agent reaches. Only once tools are reliable does orchestration become the primary lever: the 4.3%

Table 10: All harnesses under native vs. unified tools (Phases 1 and 3 combined).

Harness	Native Score	Unified Score	Δ	Planning Type
Linear Pipeline	0.843	0.970	0.127	Deterministic fixed
Adaptive Planner	0.843	0.970	0.127	Deterministic adaptive
Verify & Retry	0.870	0.970	0.100	Deterministic + recovery
LLM Planner (Claude-Opus-4.6)	0.734	0.928	0.194	Stochastic LLM

Table 11: Per-category mean score, LLM planner, native vs. unified (Phase 3).

Category	Native (mean ± std)	Unified (mean ± std)	Δ Score	n
Preprocessing	0.878 ± 0.248	0.992 ± 0.019	0.113	60
Feature Engineering	0.902 ± 0.078	0.950 ± 0.050	0.047	40
Core Analysis	0.852 ± 0.091	0.980 ± 0.040	0.128	50
Integration	0.260 ± 0.397	0.698 ± 0.376	0.438	40
Interpretation	0.675 ± 0.347	0.975 ± 0.043	0.300	40

Table 12: LLM run-to-run variance by task difficulty (Phase 3).

Difficulty	Native Variance	Unified Variance	n Tasks
Easy	0.0057	0.0000	4
Medium	0.0061	0.0000	10
Hard	0.0762	0.0116	7
Expert	0.0037	0.0000	2

Table 13: Residual orchestration gap by model relative to the rule-based optimum (0.970) (Phase 3).

Model	Unified Score	Residual Gap vs. 0.970
Claude Opus 4.8	0.963	+0.007 (near-optimal)
GLM-5.2	0.938	+0.032
Claude Opus 4.6	0.928	+0.042
Kimi K3	0.926	+0.044
GPT-5.6-sol	0.911	+0.059
GPT-5.4	0.898	+0.072
DeepSeek-V4-Pro	0.896	+0.074
Gemini-3.7-flash	0.895	+0.075

Table 14: LLM Planner (Claude-Opus-4.6) performance across tool layers (23 tasks × 10 repeats × 3 layers = 690 runs).

Tool Layer	Mean Score	Std	Error Rate	Run-to-Run Variance
Native	0.734	0.347	High (7 errors/task common)	0.0271
Unified	0.928	0.193	0%	0.0035
Unified + Validated	0.928	0.193	0%	0.0035

Table 15: Inter-harness variance summary (Full Experiment, 3 harnesses × 23 tasks × 10 repeats).

Tool Layer	Mean Variance	Max Variance	Tasks with Non-Zero Var.	Mean Score
Native	0.0025	0.0085	13/23 (57%)	0.852
Unified	0.0000	0.0000	0/23 (0%)	0.970
Unified + Validated	0.0000	0.0000	0/23 (0%)	0.970

Table 16: Per-category mean score by model (native → unified), 23 benchmark tasks (Phase 3).

Category	Opus 4.6	Opus 4.8	GPT-5.4	GPT-5.6-sol	Kimi K3	GLM-5.2	DeepSeek	Gemini
Preprocessing	0.88 → 0.99	0.88 → 0.99	0.82 → 0.98	0.88 → 0.99	0.86 → 0.98	0.86 → 0.98	0.94 → 1.00	0.93 → 0.99
Feature Eng.	0.90 → 0.95	0.88 → 0.95	0.88 → 0.95	0.88 → 0.95	0.85 → 0.95	0.84 → 0.96	0.91 → 0.95	0.91 → 0.96
Core Analysis	0.85 → 0.98	0.85 → 0.98	0.75 → 0.78	0.80 → 0.98	0.69 → 0.96	0.83 → 0.92	0.91 → 0.98	0.92 → 0.98
Integration	**0.26** → 0.70	0.85 → 0.90	0.67 → 0.79	**0.33** → 0.68	0.48 → 0.72	0.52 → 0.83	0.55 → **0.49**	0.59 → **0.50**
Interpretation	0.68 → 0.97	0.68 → 0.97	0.66 → 0.97	0.58 → 0.90	0.60 → 0.98	0.65 → 0.97	0.84 → 0.97	0.83 → 0.98

Table 17: Cross-model LLM planner performance (690 runs per model; native → unified).

Model	Vendor	Native	Unified	Δ Score	Cohen's d	Var. Red.	Mann–Whitney p
Claude Opus 4.6	Anthropic	0.734 ± 0.347	0.928 ± 0.193	+0.194	0.69	87%	1.2×10^{-23}
Claude Opus 4.8	Anthropic	0.832 ± 0.216	0.963 ± 0.047	+0.131	0.84	100%	1.4×10^{-26}
GPT-5.4	OpenAI	0.760 ± 0.297	0.898 ± 0.242	+0.138	0.51	76%	1.0×10^{-24}
GPT-5.6-sol	OpenAI	0.712 ± 0.358	0.911 ± 0.227	+0.199	0.67	79%	6.3×10^{-23}
Kimi K3	Moonshot	0.710 ± 0.357	0.926 ± 0.193	+0.217	0.76	73%	3.0×10^{-26}
GLM-5.2	Zhipu	0.755 ± 0.307	0.938 ± 0.161	+0.182	0.75	57%	7.5×10^{-29}
DeepSeek-V4-Pro	DeepSeek	0.843 ± 0.282	0.896 ± 0.264	+0.053	0.19	77%	5.3×10^{-5}
Gemini-3.7-flash	Google	0.849 ± 0.271	0.895 ± 0.272	+0.046	0.17	86%	2.4×10^{-5}

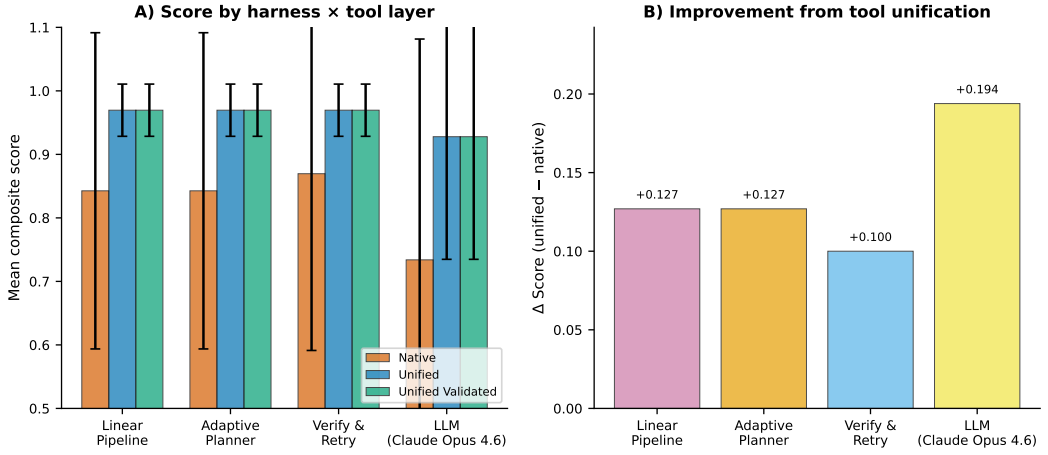

Figure 7: **All harnesses comparison.** (A) Score by harness × tool layer. LLM planner benefits most from unification (Δ = +0.194) but achieves the lowest unified score (0.928 vs 0.970). (B) Improvement magnitude: LLM > Linear = Adaptive > Verify&Retry.

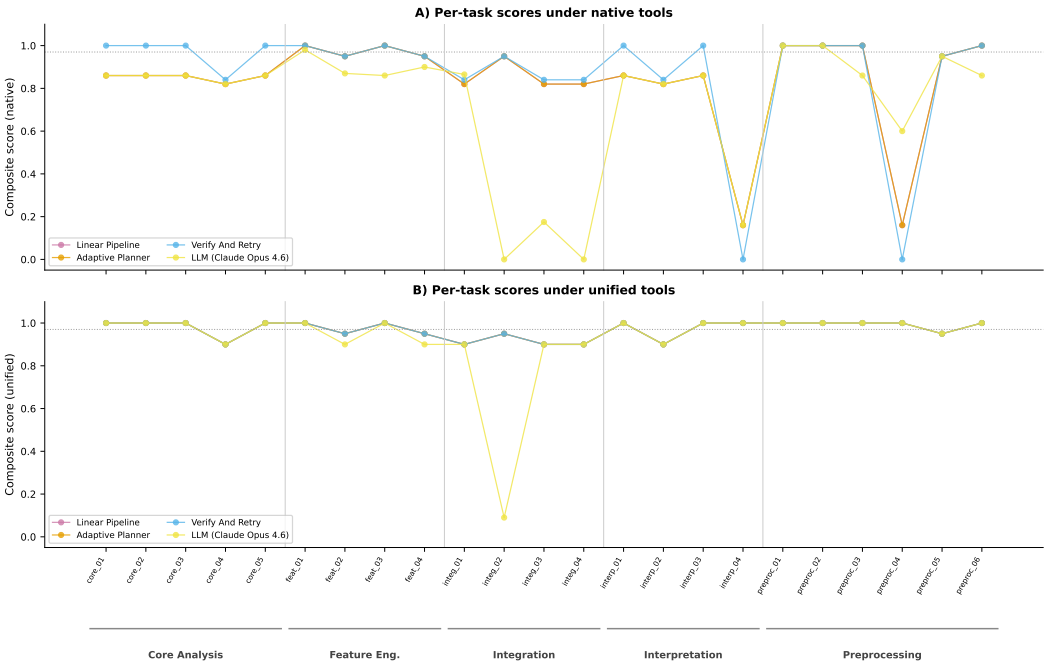

Figure 8: **Per-task score trajectories.** (A) Under native tools, harnesses diverge on specific tasks. (B) Under unified tools, all harnesses converge (overlapping lines). Dashed line at 0.97 marks the unified ceiling.

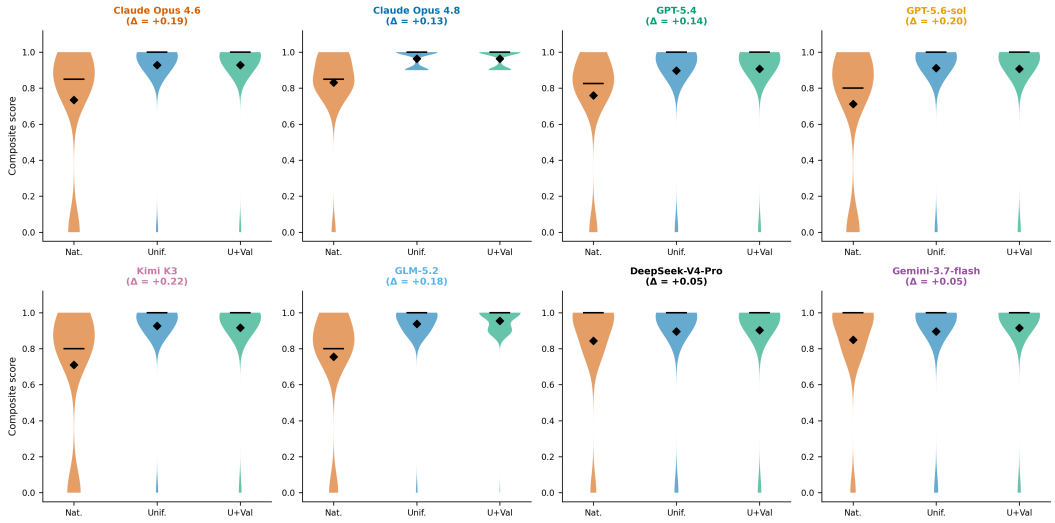

Figure 9: **Per-model composite-score distributions by tool layer (one violin panel per model), with the native → unified mean shift (Δ) annotated.** Complements Figure 2 by showing each model's full distribution rather than only summary statistics.

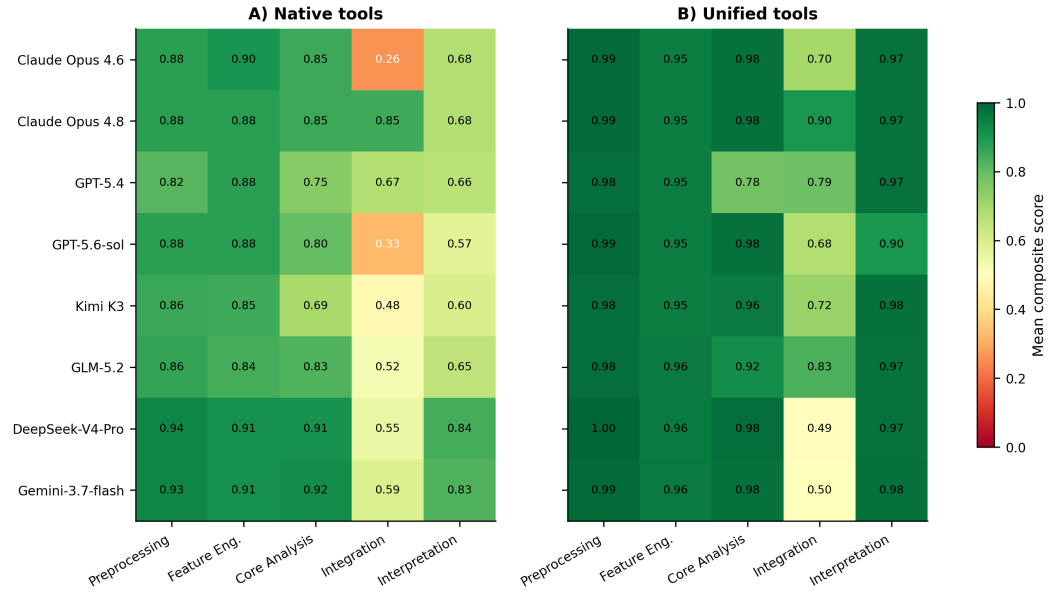

Figure 10: **Mean composite score by model × task category, under native tools (A) and unified tools (B).** The green-to-red scale highlights model-specific weaknesses: Claude Opus 4.6's native integration collapse (0.26, panel A) and the integration deficits of DeepSeek-V4-Pro and Gemini-3.7-flash that persist under unified tools (0.49 and 0.50, panel B).

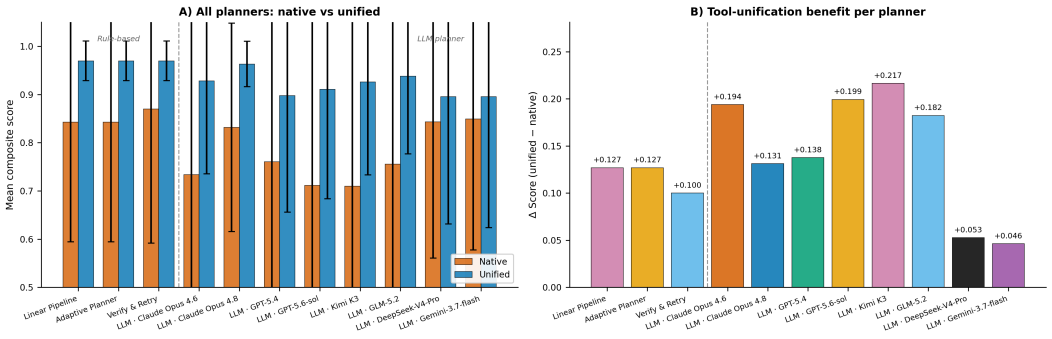

Figure 11: **All eleven planners**, the three rule-based harnesses (from the full 2,070-run experiment) and the LLM planner instantiated by each of the eight models, on a single axis. **A)** Mean composite score, native vs unified. **B)** Tool-unification benefit (Δ score) per planner.

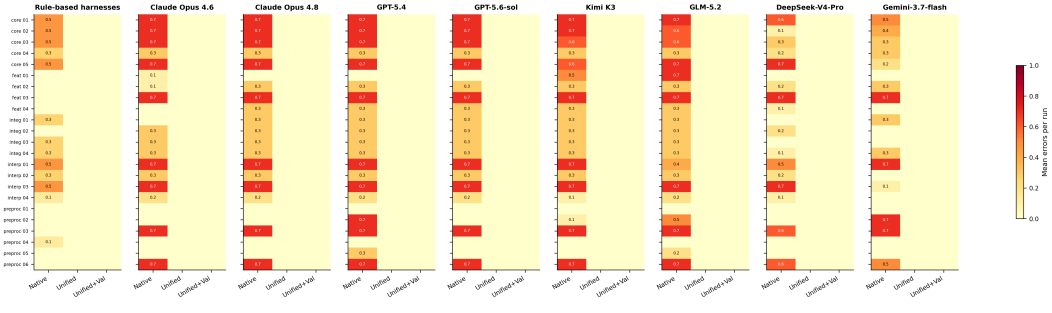

Figure 12: **Mean validation errors per run, by task (rows) × tool layer (columns). Left panel:** rule-based harnesses; **remaining panels:** each LLM. Errors are confined to the native tool layer and are eliminated under unified tools for every planner, confirming that the injected failure modes, not the planning, are the error source.

gap between Claude-Opus-4.6 (0.928) and the optimal rule-based strategy (0.970) under unified tools quantifies the remaining opportunity for planning improvement. That gap is real but small, suggesting that current LLMs are already adequate planners for well-defined sc-RNA pipelines and that further gains require either harder tasks or qualitatively different planning capabilities such as creative experimental design and novel method selection. Validators, finally, belong as defense-in-depth rather than as the primary quality mechanism: they provide essential safety guarantees when tools occasionally fail in production (through edge cases, version mismatches, or novel data distributions not seen during development), but the Phase 2 null result argues against relying on them as a substitute for tool reliability, not against validators as such.

These findings carry a direct policy implication for the bio-agent community: shared, validated tool registries—analogous to Bioconductor (Huber et al., 2015) for R or Scanpy (Wolf et al., 2018) for Python—should be developed as community infrastructure for AI agents. Each agent project currently implements its own tool wrappers, often reintroducing the same errors we inject in native mode, such as double normalization and batch ignoring. A shared registry with validation guarantees would eliminate a major source of inter-agent variance (per our Phase 1 result), enable meaningful comparison of planning strategies on equal footing, reduce duplicated engineering effort across projects, and provide a quality floor that benefits every agent. The MCP Bridge in Dissect-Audit (§C.11) offers a technical template for how such a registry could be exposed: tool definitions in a standard protocol, callable by any agent regardless of its LLM backbone or planning architecture.

Our work also suggests that current agent benchmarks partially measure the wrong thing. When BiomniBench reports that one agent outperforms another, the difference may stem primarily from tool-layer quality—which is straightforwardly fixable—rather than from planning intelligence, which requires fundamental AI advances. Benchmark designers should therefore consider offering a standardized-tools track in which agents share a common tool layer, isolating planning contributions; reporting inter-condition variance alongside mean performance to enable causal attribution; and

categorizing failure modes so that tool-induced failures (fixable via a registry) are distinguished from strategy-induced failures (requiring architectural innovation).

F.2 The Validator Paradox and Planning Stochasticity

The Phase 2 null result—validators add nothing on clean tools—creates an apparent paradox. If validators are most needed when tools fail (native mode), but there they can only detect errors *after* the data is corrupted, what is their utility? The resolution distinguishes three deployment modes. In *post-hoc auditing* (our Phase 1 use) validators are a measurement instrument, not an intervention, precisely how we quantify native-mode damage. In *intermediate checking with retry* (verify-and-retry) they check after each step and re-execute, affording partial recovery (pass rate 95.7% vs. 73.5%) but unable to undo damage from a tool that always double-normalizes. In *preventive gating* (the `GatedRegistry` of Dissect-Audit) they check inputs *before* execution and refuse to proceed, active prevention on unreliable tools. The recommendation follows: deploy validators as intermediate-checking defense-in-depth, a complement to reliable tools (+2.7% score, +22pp pass rate) rather than a substitute, since tool reliability itself adds far more (+12.7% score, +26.5pp pass rate).

Phase 3 shows why: under unreliable tools, planning stochasticity *amplifies* tool errors (different plans trigger different error combinations), whereas under reliable tools the same stochasticity is *harmless* (all plans reach correct outcomes by different paths). Two implications follow. Temperature tuning matters less than tool quality, since even at temperature = 0.7 the LLM agent produces consistent outcomes (variance ≈ 0) on clean tools; temperature governs variance only when tools are unreliable, so planning diversity is essentially free on clean tools and could even benefit creative tasks. The one exception is integration (mean 0.675, std 0.395 even under unified tools): it demands *planning correctness* (the agent must include batch-integration steps and avoid over-correction), capturing the planning bottleneck in its purest form.

G Limitations

Several limitations bound generalizability. *Simulated tool heterogeneity*: the native layer injects calibrated errors at fixed rates rather than reproducing real API differences; it captures the *types* of failures documented in BiomniBench evaluations but perhaps not their exact production *distribution*. The qualitative finding should hold for any error distribution yielding measurable native-mode variance; only the quantitative results (100% reduction, $p = 1.22 \times 10^{-4}$) depend on injection rates. *Single domain*: all experiments are sc-RNA-seq; generalization to less-structured domains (drug discovery, protein design) is uncertain, though our five-category taxonomy spans data cleaning, statistical and ML-based analysis, batch integration, and text generation. *No external agent integration*: all harnesses are built-in; the MCP Bridge supplies the infrastructure to expose the registry to external agents (Biomni, K-Dense) in follow-up work. *Sparse ground truth*: only 2 of 23 tasks carry external ground truth, but both show the same native < unified pattern as validator-scored tasks. *LLM API non-determinism*: Phase 3 numbers are a snapshot at one model version and may drift, preserving the directional finding. *Score ceiling*: the 0.970 unified ceiling (the 23-task mean) falls short of 1.0 because of batch-effect warnings on the multi-batch Kang2018 tasks where no harness plans integration, a shared planning limitation manifesting as uniformly reduced scores, not inter-harness variance.