AUTOBIOLAB: FROM SCIENTIFIC INTENT TO MACHINE EXECUTABLE LABORATORY AUTOMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

While large language models (LLMs) can increasingly translate scientific requests into instrument commands, generating a sequence of executable commands does not guarantee the intended experimental state. We study this gap in automated liquid handling, where intermediate mixtures may be transferred, diluted, and reused across multiple steps, making later operations to depend on the material changes produced earlier. To bridge LLM reasoning with precise physical realities, we introduce AutoBioLab, an agentic system that fundamentally decouples the verification of scientific intent from robotic execution. By pairing LLM-driven planning with deterministic solvers, material state replay, separate scientific and robotic contracts, this agentic architecture convert abstract experimental designs into a verifiable physical state before any action is taken. Empirically, this decoupled approach more than doubles the end-to-end planning success rate across multiple state-of-the-art LLMs, raising the number of fully correct plans from 21.3 to 44.0 out of 48, which enhance the open-source model to approximate the SOTA performance of current commercial model. Furthermore, our runtime agent improves the successful recovery of human injected errors from 66.3% to 82.3%. Ultimately, AutoBioLab demonstrates that separating what an experiment should achieve from how a robot must move is essential for reliable, machine-executable laboratory automation.

1 INTRODUCTION

Scientific agents are increasingly able to translate natural language experimental requests into actions that can be executed by laboratory instruments. Coscientist and ChemCrow connect language models with scientific tools and experimental workflows, while autonomous laboratories demonstrate the value of linking computation to physical outcomes (Boiko et al., 2023; Bran et al., 2024; Szymanski et al., 2023). This progress raises a fundamental question for end-to-end laboratory automation: **How should an agent turn a scientific goal into a sequence of hardware operations whose consequences satisfy that goal?**

We study this question in automated liquid handling, a common component of biological laboratory automation (Cheng et al., 2026). A liquid handler executes device-specific commands that move measured liquid volumes among tubes, reservoirs, and multi-well plates. For an agent, however, specifying those commands also requires deciding which materials should be prepared, where they should be placed, and how transfers connect them. Because intermediate mixtures can be reused in later steps, each transfer changes the state of the material on which subsequent operations depend. The agent must therefore plan not only executable actions but also the quantitative consequences of those actions.

Existing systems address different parts of this problem. CLAIRify, AutoLabs, and BioProAgent translate scientific requests into structured or executable procedures and use external tools or feedback to improve the generated plans (Yoshikawa et al., 2023; Panapitiya et al., 2026; Liu et al., 2026). Laboratory frameworks such as PyLabRobot already provide device abstraction and track resources, tips, liquids, and volumes (Wierenga et al., 2023). Liquid-handler planning methods can also optimize action sequences once the laboratory state and desired final state are specified (Ferdosi et al., 2023). What remains is to connect these pieces from the original scientific request: construct

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

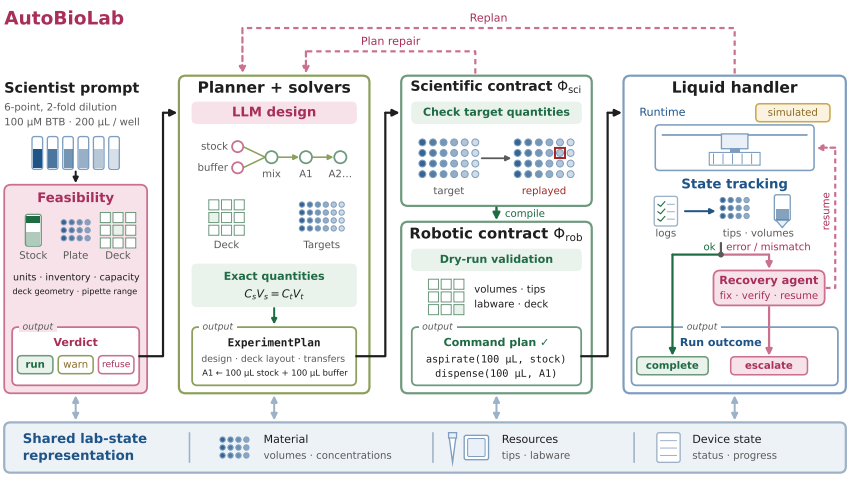

Figure 1: **AutoBioLab workflow for liquid handling planning and simulated recovery, illustrated with a six-point, two-fold serial dilution.** An LLM proposes the plan, deterministic solvers resolve quantitative dependencies, and scientific and robotic contracts verify material outcomes and device actions. A simulated runtime tracks laboratory state and supports recovery, replanning, or stopping under injected faults.

the required material state, resolve quantities that depend on earlier transfers, and verify both the resulting experimental state and the executable device actions.

AutoBioLab addresses this gap through complementary scientific state and robotic contracts. The scientific state contract checks target wells, volumes, concentrations, and resource constraints, while the robotic contract checks the compiled device actions. An LLM proposes the experimental layout and transfer structure, deterministic solvers resolve supported quantitative dependencies, and material state replay computes the consequences of the planned transfers. This separation also guides repair. Numerical inconsistencies can be solved again, replayed outcomes that are mismatched with scientific requirements trigger plan revision, and invalid device commands can be corrected independently. Figure 1 summarizes this workflow from a scientific request to validated liquid handling actions.

We evaluate AutoBioLab in the planning and simulated recovery. BTB-48 tests whether the generated liquid-handling plans satisfy the target material states, resource constraints, and device requirements for 48 tasks. Across four tested LLM backbones, AutoBioLab increases planning success; with Sonnet 5, the mean number of successful tasks rises from 21.3 to 44.0 of 48. Controlled ablations separately examine quantitative solving, scientific check and repair, and robotic command repair. We also inject device, material, and observation faults during simulated execution. A recovery is counted as correct when the runtime restores the corresponding fault-free material state when possible, or stops appropriately when recovery is not possible.

In summary, this paper makes three contributions:

- **Science intent driven machine executable lab automation agent.** We introduce AutoBioLab, a novel agent framework that pairs executable LLM planning that satisfy both scientific and robotics real-world requirements and enable highly targeted, bounded error repair.
- **A verifiable state-of-art benchmark for state-aware planning and closed-loop recovery.** We construct BTB-48, a rigorous benchmark of 48 liquid-handling tasks scored by deterministic simulation rather than subjective LLMs, alongside a 96-scenario testbed to evaluate runtime fault recovery.
- **Systematic validation on agent performance with promising results** Comprehensive evaluations of four LLMs demonstrate that more than doubles the end-to-end plan success rate (from 21.3 to 44.0 out of 48 tasks with Sonnet 5) and improves successful physical error

recovery(from 66.3% to 82.3%), significantly demonstrating that open-source model successfully approximate the SOTA performance of commercial model with our agent framework.

2 METHODS

AutoBioLab translates abstract experimental requests into executable liquid-handling programs by bridging the gap between language-model reasoning and material-state physics. Rather than relying on a single generation step, our system decouples the workflow: the language model proposes the experimental structure, deterministic solvers compute the precise quantities, and scientific checks verify the outcome against the user's intent. If discrepancies appear, a bounded repair module will review the plan and then it will be compiled to robot commands list and tested by robot checker.

2.1 PROBLEM FORMULATION AND SEMANTICS

To successfully complete the experiment automatically by our agent, the system must first fit the natural language request x and the hardware configuration H into a trackable physical reality. Therefore, our goal is to construct a program that satisfies the final requested material state while respecting all resource constraints. The planner infers the initial content of the request and binds them to the labware. That is the initial state m_0. In this way, we can obtain the intermediate plan which contains a design, a deck layout, an operation graph, quantitative parameters, and declared terminal results G_{plan}.

While the device state tracks equipment, tips, and command progress, the material state strictly tracks liquid composition. Let $v_{w,r}$ denote the volume contribution of original liquid r in well w, with source concentration c_r. Assuming ideal homogeneous mixing, the total volume, solute quantity, and the concentration are given by:

$$V_w = \sum_r v_{w,r}, \qquad q_w = \sum_r c_r v_{w,r}, \qquad c_w = q_w / V_w \quad (V_w > 0). \tag{1}$$

When a transfer of volume δ moves from source s to destination d, it updates both wells proportionally:

$$\Delta v_r = \delta \frac{v_{s,r}}{V_s}, \qquad v' * s, r = v * s, r - \Delta v_r, \qquad v' * d, r = v * d, r + \Delta v_r. \tag{2}$$

Additionally, liquid handling is highly order-sensitive. Transferring liquid from a well before adding a buffer yields a vastly different final concentration than transferring the same volume after dilution, even if the terminal volumes look identical. Therefore, the system must simulate the full compositional trajectory.

For commands $P = (a_1, \ldots, a_T)$, replay computes the state transition $m_t = F(m_{t-1}, a_t)$. Ultimately, we define a scientific state contract that enforces both the final goal and the valid physical trajectory to get there:

$$\Phi_G(P; m_0) = G(m_T) \wedge \bigwedge_{t=1}^{T} I(m_{t-1}, a_t, m_t). \tag{3}$$

Here, constraints such as target volumes, sufficient source liquid, and container capacity are continuously checked. This equation defines the desired semantics; our implementation checks supported subsets of these conditions rather than attempting an exhaustive verification of Φ_G.

2.2 JOINT QUANTITATIVE SOLVERS

Using the proposed experimental structure, deterministic solvers compute precise volumetric parameters. Local passes handle direct dilutions (e.g., $v_s = c^\star V^\star / c_s$ for empty destinations), while joint passes resolve complex interdependencies caused by downstream withdrawals.

For a serial dilution chain without terminal discard—where well i retains volume $V_i^\star$ at concentration c_i and transfers u_i to $i+1$—the incoming transfer and zero-concentration buffer prefill satisfy, for $i = 2, \ldots, n$:

$$u_{i-1} = \frac{c_i}{c_{i-1}}(V_i^\star + u_i), \qquad b_i = V_i^\star + u_i - u_{i-1}, \qquad u_n = 0. \tag{4}$$

To satisfy these coupled constraints, the solver propagates backwards via substitution, requiring $V_1^\star + u_1$ for the initial well. Because this synthesis relies strictly on topological transfer edges rather than rigid task labels, all proposed updates are rigorously verified by composition replay, maintaining state-aware consistency even if external feedback loops are disabled.

2.3 Task checking and bounded repair

Basic internal checks (e.g., volume balance, local concentrations) cannot detect plans that perfectly execute an incorrect goal. To bridge this gap, a separate model extracts per-well targets $\widehat{G}_{\mathrm{req}}$ directly from the prompt to validate the final results. Moreover, this extraction never accesses the true evaluation reference $G^\star$, preserving benchmark integrity. Upon detecting discrepancies, AutoBioLab first attempts deterministic re-solving to update quantities against $\widehat{G}_{\mathrm{req}}$. If structural flaws persist, it triggers up to two model rewrites using diagnostic reports. To prevent regressions, candidate plans are strictly gated by a violation-count vector, ensuring monotonic improvement across internal checks. Ultimately, the system proposes a plan, applies requirement-guided re-solving, rewrites the graph if necessary, and forwards the best candidate to compilation.

2.4 Robotic compilation and localized repair

The verified scientific blueprint is deterministically lowered into executable commands, resolving physical specifics like pipette selection and stroke splitting. Structural dry runs flag command-level failures, triggering up to three localized, model-driven command patches. Because robotic and scientific repairs operate on fundamentally separate representations, fixing a physical collision does not alter the underlying chemistry. Restoring physical executability does not establish scientific correctness, necessitating independent final evaluation.

2.5 Stateful execution and verified recovery

Commands execute via an Opentrons HTTP session (Opentrons Labworks, 2026) that continuously tracks historical progress and material states. For runtime anomalies, a ReAct-style controller (Yao et al., 2023) uses session context to perform bounded, context-aware corrections—such as safely returning aspirated liquid to its source—avoiding blind full-protocol restarts. Any claimed recovery is strictly verified by re-executing the failed step or visual checkpoint before the protocol is allowed to continue.

3 Experimental Setting

3.1 Datasets and tasks

We build the BTB dataset from liquid handling requests using bromothymol blue and a buffer. Each request includes a ground truth reference specifying target volumes and concentrations. The main BTB 48 dataset contains 48 tasks that span four families, three difficulty levels, and four instances. These families are: *normalization* (concentration and initial states), *resource constraints* (supply and capacity limits), *serial dilution* (dependent transfers), and *multistage preparation* (cross plate scheduling).

All systems are evaluated as black boxes. The *benchmark and overall ablation* use BTB 48 unchanged. API models generate plans directly, while agent systems run their full pipelines. *Mechanism ablations* use 12 separate dilution chain tasks that specify layouts and concentrations but withhold transfer quantities. For *recovery*, we use 14 pre corrected BTB 48 plans to build 96 fault scenarios per round. A simulator injects single, resource, and persistent errors during execution. The system must observe this feedback to correct, resume, or stop the process. The robot and camera endpoints are simulated. The Appendices A, B and C detail the task construction.

3.2 Evaluation criteria and scoring

We score all outputs using deterministic programmatic checks against the reference outcomes. We do not use LLM judges.

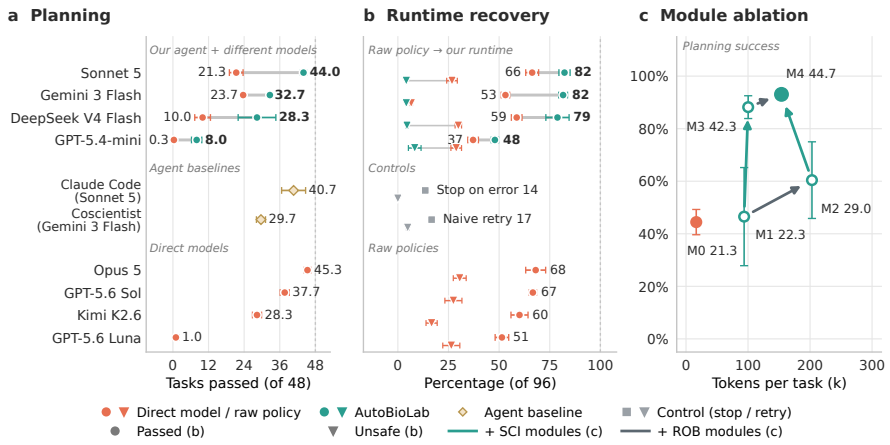

Figure 2: **Planning, runtime recovery, and overall ablation. a:** BTB-48 planning success, three-round means ± sample standard deviation; connected points compare the same backbone. Success requires Structure, Science, and Bounds jointly. **b:** Runtime recovery of 96 fault scenarios; three-round means ± sample standard deviation. Circles show correct handling. Triangles show the percentage of scenarios with at least one unsafe action over all (lower is better). **c:** M1–M4 with neither group, ROB, SCI, or both; M0 is the direct Sonnet reference. Coordinates are three-round means of token use and planning success; vertical bars show success SD. Appendix Tables 5, 17, 7 and Table 17 give full scores.

For the *benchmark and the overall ablation*, a task passes only if the plan satisfies structural validity, all target volumes and concentrations, and hardware bounds. The evaluator runs the plan in the simulator and compares the final state with the reference. Any failed component or missing plan counts as a failure. We report complete plan success alongside per component scores. *Mechanism ablations* also report detector recall, false alarms, and repaired plans.

The 96 *recovery* tasks include 83 recoverable cases and 13 handover cases. Recoverable cases must achieve the fault free reference state. Handover cases must be performed safely without executing unsafe actions. The Appendices A.2 and C specify tolerances and scoring functions.

4 Results

We perform a planning benchmark, ablation study, module mechanism test, and also have a case study for each module. The simulated runtime recovery result appears in Appendix D.

4.1 Planning benchmark

We compare AutoBioLab, direct API models, and two agent baselines on the same BTB-48 dataset. A task passes only when its plan satisfies structural validation, reaches the requested well volumes and concentrations, and respects resource bounds. All these processes will be scored by the deterministic examiner in benchmark workflow. Figure 2a reports the mean number passed across three rounds, with error bars showing sample standard deviation.

AutoBioLab improves complete-plan success across all four tested backbones. The clearest gain is on Sonnet 5, where the mean of success tasks rises from 21.3 to 44.0 of 48, more than doubling the number of correct plans. With the same backbone, AutoBioLab also achieves higher mean success and lower observed run-to-run variation than Claude Code. And we also use the low level model(Sonnet 5) with the help of our agent workflow to reach the SOTA level model(Opus 5) performance, 45 of 48. These results support the value of structured planning and verification across the tested models that gain the highest mean success.

The component scores reveal why complete-plan correctness is challenging. Direct Sonnet often achieves the target well states while failing command validation. Thus, command validity alone

is insufficient to establish experimental correctness. Instead, successful planning must satisfy both quantitative outcomes and execution constraints. This motivates AutoBioLab's combined scientific and robotic checks and the following ablations examine their individual contributions. Detailed scores and costs appear in the Appendix Tables 5 and Appendix A.4.

4.2 Runtime recovery

To evaluate recovery capabilities, we inject device, material and observation faults into 14 BTB-48 plans, generating 96 test scenarios per round. A trial is considered successful only if it achieves the original fault-free state or safely stops when physical recovery is impossible, as shown in Figure 2b.

AutoBioLab's runtime improves performance across all backbones. Most importantly, the framework enables an open-weight model to surpass the performance of standalone closed-source models. After deploying DeepSeek V4 Flash within AutoBioLab, its success rate increases from 58.7% to 78.8%, beating both closed-source alternatives, Opus 5 at 68.1% and GPT-5.6 Sol at 66.7%. And it also has the same view for Sonnet 5. This robust recovery requires approximately three times the model calls per episode. The complete results are detailed in the Appendix D.

4.3 Overall ablation: scientific and robotic module groups

We independently enable two module groups in a common Sonnet 5 planning skeleton: scientific (S) computes quantities, checks outcomes, and repairs plans while robotic (R) validates and repairs commands. The four combinations form (M1–M4) evaluated on the same requests with the same joint pass criterion as the benchmark. Figure 2b shows planning success, while Appendix Table 7 provides component scores and token use.

The scientific group provides the largest improvement: enabling S raises mean success from 22.3 to 42.3 of 48 tasks without robotic support. This performance increase shows the necessity of scientific module within the configured pipeline. After introducing the robotic module, it increases success to 44.7 which is the highest among the variants. It suggests that the robotic module also contributes to check and fix some types of comment errors which will be discussed on Chapter 4.5 Case Study.

The science-only variant uses 34.9% fewer tokens than the entire system while retaining most of its success, exposing the cost of additional checking and repairing. These comparisons evaluate configured groups: the prompts delegate calculation to the solver, so directly closing S stops concentration and volume computation. Compiler and budget differences across rounds further limit attribution (Appendix A.5). The following experiments isolate the detailed contributions for both SCI and ROB modules.

4.4 Module ablation: scientific and robotic mechanisms

To evaluate the contribution of individual mechanisms, we conducted an ablation study by fixing either the model replies or the stored plans. For science module, we have three parts: joint solver, scientific check and plan repair; For robot module, we also have error detection and located repair which is tested as one complete sub-module. In detail, we assessed each module using metrics tailored to its specific function: complete-plan success for the solver and scientific repair modules, detection rate for the checkers; and both correction success rate and the number of model calls per correction for the robotic repair module.

Scientific module: coupled quantities and task-derived targets. Our first main finding demonstrates that joint solver and task-derived verification can improve planning accuracy and error detection. After reasoning about the plan, the joint solver resolves downstream dependencies that local, step-by-step arithmetic fails to capture. When model replies are kept constant, the joint solver increases the mean complete-plan success rate from 2.5 to 9.2 (Figure 3a), while the per-step solution does not yield improvement. This supports the assumption that downstream constraints such as final transferring states are taken into account, rather than simply calculating individual dilution ratios correctly.

Furthermore, our results also demonstrate that scientific checks and quantitative solver play essential roles in automated plan correction. As shown in Figure 3b, pairing full verification with active

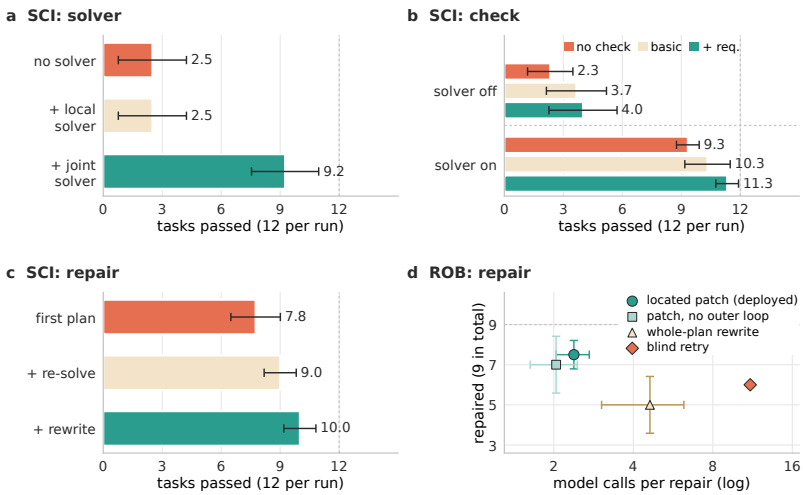

Figure 3: **Scientific (SCI) and robotic (ROB) mechanism ablations on three repetitions of 12-task sets.** Means with standard deviation. **a:** Complete-plan success: *no solver* leaves quantities unchanged; *local solver* sizes additions independently; *joint solver* coordinates quantities across linked transfers. **b:** Complete-plan success. Each initial plan enters *no check*, *basic* checks (coverage, balance, and self-consistency), and + *req.* (additional task-derived per-well target checks), with joint solvers off/on. Active checks gate a common bounded repair policy; no check performs no scientific repair. **c:** Complete-plan successes through scientific repair. First plan means just accept original plan and no checking; re-solving uses no model calls; rewriting uses eight and six calls across the three experiments. **d:** Three repair repetitions on the same nine robotic failures, which are discovered in 12-task sets. *Located patch* edits diagnosed commands with detailed error information from checkers; *blind retry* only throughout error signal to retry but no error details for models; *whole-plan rewrite* ask model to regenerate all commands with the same information as the Located patch group; *patch, no outer loop* removes the recompilation loop.

solvers increases the mean success rate from 9.33 to 11.33 out of 12. Within this setup, advanced requirement checks successfully caught three subtle errors that are higher than basic validation and resolved them deterministically without any extra LLM calls. In contrast, removing solvers plummets the success rate to just 4.0 out of 12, illustrating that diagnostics alone cannot fix complex planning failures. This synergy remains consistent across separate historical traces, where applying requirement-aware rewriting and re-solving improved success from 7.75 to 10.0 out of 12 (Figure 3c). Ultimately, these findings confirm that robust plan correction requires tightly coupling strict verification with quantitative solvers.

Robotic module: error localization and bounded command repair. We evaluated four repair strategies on nine model-generated robotic failures. These errors emerged from otherwise valid plans and were reliably caught by physical boundary checks; without intervention, they lead to terminal execution failure, making a "no-repair" control unnecessary. Starting from identical failure states, we compared our deployed localized patch (specific diagnostics, targeted edits) against three baselines: blind retry (no error text provided), whole-plan rewrite (diagnostics provided, full regeneration), and a patch without the outer loop.

Our localized patch module achieves superior repair efficiency and token utilization in all baselines (Figure 3d). Guided by concrete diagnostics, our approach repairs 7.5 out of 9 failures while requiring only 2.4 model calls per correction. In contrast, blind retry fixes only 6.0 errors while demanding 11.1 calls. Since this control inadvertently retained some outer-loop context, a fully blinded approach would perform even worse, proving that omitting specific feedback severely wastes computational budget. Developing a tight scope for these edits is equally critical. Rewriting of the Whole-plan drops the success rate to 5.0 out of 9, as full regeneration often breaks valid steps and hits output limits. Additionally, disabling the outer verification loop yields 7.0 repairs at 2.0 calls

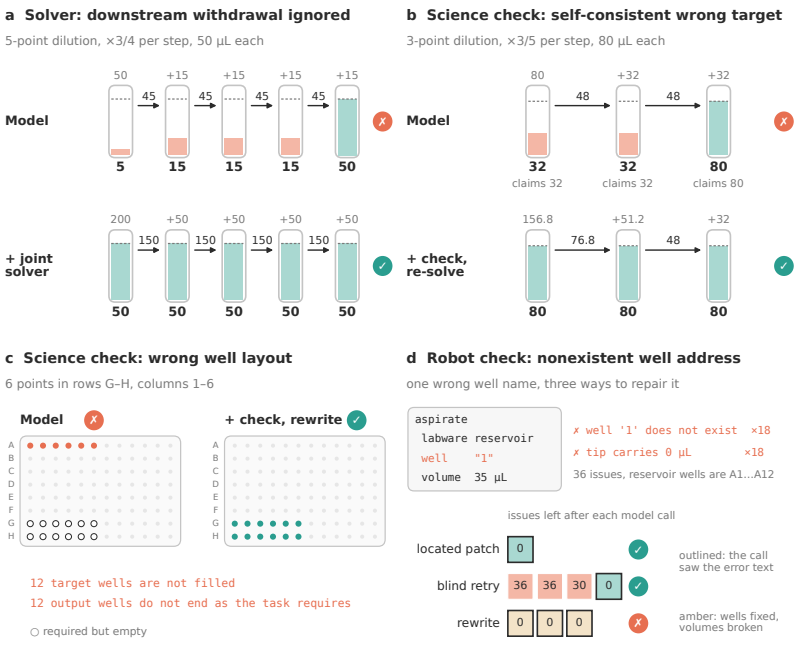

Figure 4: **How the mechanisms correct planning errors. a:** Joint synthesis accounts for liquid withdrawn by later steps. **b:** Task-requirement checks expose a plan that agrees with its own incorrect target and trigger re-solving. **c:** Supplying the required layout guides a structural rewrite. **d:** Diagnostics identify invalid reservoir addresses; localized patches fix them while preserving the recipe. Outlined boxes mark calls receiving error text; amber boxes mark corrected addresses with incorrect volumes.

each, showing no substantial benefit from full-plan recompilation. Ultimately, combining precise error context with bounded modifications maximizes both correction success and token efficiency.

4.5 Case study results

We track four stored plans from error to correction. These cases explain the mechanisms behind the results of sub-modules listed before and also explain the reason for their higher performance.

Correct ratios can still produce incorrect final volumes. The model chooses the correct dilution ratios, but overlooks that each prepared well supplies the next (Figure 4a). The loading of the first well with its final target of 50 μL leaves only 5 μL after withdrawal. Local solving cannot fix this dependency because the ratios are already correct. Our reviewer found that the joint solver coordinates upstream loading, transfers, and buffer additions across the chain. This module makes sure that each well retains its target after supplying its successor.

A plan must agree with the task as well as with itself. The second plan declares the under-filled state that its commands produce: 32 μL in wells that require 80 μL (Figure 4b). The results of commend replaying in the simulator agree with that declaration, so self-consistency checks remain silent. After reviewing the model trajectories, we found that comparing to independently extracted task targets will reveal the planning mismatch. The transfer structure is usable, allowing the solver to recompute quantities against the correct targets without a model rewrite and restore the requested volumes and concentrations.

A structural error needs the missing requirement, not just new quantities. The design compresses two requested rows into A1–A6. It seems correct but leaves the intended wells in G–H empty (Figure 4c). Re-solving amounts cannot create the missing outputs. After the checker identifies the absent target wells, the repair prompt supplies their required layout and end states. Our reviewer found that based on our scientific repair module, the model rebuilds the design around all

12 wells and the solver resolves its quantities into a passing plan. While an earlier method receiving only error messages retained the wrong layout.

Concrete diagnostics focus command repair. The invalid reservoir addresses block aspiration, causing errors in the missing-well and empty-tip (Figure 4d). A localized patch uses the diagnostic to change `1/2` to `A1/A2`, preserving the recipe and solving all 36 issues in one call. Blind retry instead revises unrelated arithmetic until an outer-loop call reveals the error. Whole-plan rewriting fixes the addresses but changes valid volumes and fails within budget. Our discovery for this case is identifying the cause and restricting the edit preserves what already works.

5 Related Work

Scientific agents and laboratory interfaces. Autonomous experimentation spans robot scientists, self-driving synthesis laboratories, and agents for experimental design (King et al., 2009; Szymanski et al., 2023; Roohani et al., 2025). Coscientist, ChemCrow, and ORGANA connect language-model reasoning to laboratory tools and execution (Boiko et al., 2023; Bran et al., 2024; Darvish et al., 2025). AutoLabs combines quantitative tools with iterative self-correction, while BioProAgent combines scientific review with deterministic physical-constraint checking and repair. (Panapitiya et al., 2026; Liu et al., 2026). Chemputer/XDL and CLAIRify show how experimental procedures can be represented in machine-readable form and translated toward robotic execution (Mehr et al., 2020; Yoshikawa et al., 2023). At the interface level, PyLabRobot provides a common interface for liquid-handling devices and tracks tips, resources, liquids, and volumes (Wierenga et al., 2023). More recently, Anthropic's Model Hardware Standard (MHS) research preview proposes a shared interface for agents to discover devices, read device state, and issue supported operations (Anthropic, 2026). AutoBioLab contributes an experiment-level material-state representation and quantitative planning mechanisms above its device adapter.

Planning, verification, and runtime recovery. Prior work has addressed executable protocol generation, formal representations of sample state, and liquid-handler action planning from specified laboratory states (Whitehead et al., 2018; Cardelli et al., 2021; Ferdosi et al., 2023). More recent work introduces compiler-tracked or evolving laboratory state together with deterministic checking and simulation (Song et al., 2026; Li et al., 2026). Separating model generation from external computation and checking is also related to PAL and LLM-Modulo (Gao et al., 2023; Kambhampati et al., 2024). Recent laboratory systems have begun to extend these ideas into robotic execution. PRISM uses digital-twin simulation to refine protocols before execution (Hsu et al., 2026), AEGIS combines assay-aware validation with visual runtime monitoring of liquid-handling failures (Setty et al., 2026), and RoboChemist uses visual feedback and re-execution during long-horizon robotic chemistry (Zhang et al., 2025). AutoBioLab's recovery controller follows the reasoning-action-observation pattern of ReAct (Yao et al., 2023), but maintains material and device state across an interrupted run and verifies corrective actions before continuing. Our recovery experiments evaluate these behaviors under controlled simulated device, material, and observation faults. BTB-48 additionally complements BioPlanner and LAB-Bench by evaluating generated liquid-handling plans against explicit per-well quantitative targets and resource constraints (O'Donoghue et al., 2023; Laurent et al., 2024).

6 Discussion and Limitation

AutoBioLab demonstrates the power of framing laboratory actions as material state transformations. By pairing joint quantitative solving with distinct scientific and robotic checks, our system effectively isolates protocol flaws from execution errors to guide precise repairs. Moreover, our findings highlight that internal self-consistency is insufficient; verification must be grounded in task-derived targets to prevent a system from flawlessly executing the wrong experiment. While simulated recovery results underscore the value of integrating state tracking and observation, these benefits are inherently bounded. Future efforts must bridge the gap to real-world deployment through prospective physical experiments, emphasizing uncertainty-aware state estimation, independent task specifications, and explicit stopping criteria. Ultimately, while generalized reliability remains an open challenge, AutoBioLab establishes a practical foundation to ensure executable actions achieve targeted experimental states.

7 REPRODUCIBILITY STATEMENT

All codes for the Autobiolab agent architecture will be published on Github after being accepted by the conference. Appendix A describes planning controls, and Appendix C records recovery splits, policy definitions, metrics, and disclosed harness fixes. All experiment hyperparameters for each module are listed at the beginning of the Appendix.

8 AI USING STATEMENT

This paper is written by humans with the help of generative AI (ChatGPT, Claude, Gemini), mainly focusing on academic writing, scientific illustration and data visualization. The scientific problem and the main solution are discovered and designed entirely by humans.

REFERENCES

Anthropic. Previewing the Model Hardware Standard. https://www.anthropic.com/news/model-hardware-standard-research-preview, August 2026. Research preview announcement, August 27. Accessed 2026-09-07.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624:570–578, 2023. doi: 10.1038/s41586-023-06792-0.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6:525–535, 2024. doi: 10.1038/s42256-024-00832-8.

Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. A language for modeling and optimizing experimental biological protocols. *Computation*, 9(10):107, 2021. doi: 10.3390/computation9100107.

Wenduo Cheng, Mingqian Ma, Shuaike Shen, Anna Hupalowska, Jennifer E. Rood, Yang Zhang, Gaurav Agrawal, Christine Bakan, Michelle A. Lee, Aviv Regev, and Jian Ma. Towards human-led, agent-driven autonomous laboratories for the life sciences. *Preprints*, 2026. doi: 10.20944/preprints202608.0273.v1.

Kourosh Darvish, Marta Skreta, Yuchi Zhao, Naruki Yoshikawa, Sagnik Som, Miroslav Bogdanovic, Yang Cao, Han Hao, Haoping Xu, Alán Aspuru-Guzik, Animesh Garg, and Florian Shkurti. ORGANA: A robotic assistant for automated chemistry experimentation and characterization. *Matter*, 8(2):101897, 2025. doi: 10.1016/j.matt.2024.10.015.

Mohsen Ferdosi, Yuejun Ge, and Carl Kingsford. Reinforcement learning for robotic liquid handler planning. In Djamal Belazzougui and Aïda Ouangraoua (eds.), *23rd International Workshop on Algorithms in Bioinformatics (WABI 2023)*, volume 273 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 23:1–23:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPIcs.WABI.2023.23. URL https://doi.org/10.4230/LIPIcs.WABI.2023.23.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *PMLR*, pp. 10764–10799, 2023.

Brian Hsu, Priyanka V. Setty, Rory M. Butler, Ryan Lewis, Casey Stone, Rebecca Weinberg, Thomas Brettin, Rick Stevens, Ian Foster, and Arvind Ramanathan. Prism: Protocol refinement through intelligent simulation modeling. *Digital Discovery*, 5(6):2613–2628, 2026. doi: 10.1039/D6DD00004E.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B. Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pp. 22895–22907, 2024.

Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The automation of science. *Science*, 324:85–89, 2009. doi: 10.1126/science.1165620.

Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. LAB-Bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024. doi: 10.48550/arXiv.2407.10362.

Xiaobo Li, Luyao Ge, Xiaohui Li, Lulu Guo, Ming Mao, Jiwang Zheng, Wenting Guan, Xin Yang, Yi Luo, Jun Jiang, and Linjiang Chen. A computable representation of the physical laboratory enables verifiable workflows, 2026.

Yuyang Liu, Jingya Wang, Liuzhenghao Lv, and Yonghong Tian. BioProAgent: Neuro-symbolic grounding for constrained scientific planning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 42764–42783, 2026. doi: 10.18653/v1/2026.acl-long.1981.

S. Hessam M. Mehr, Matthew Craven, Artem I. Leonov, Graham Keenan, and Leroy Cronin. A universal system for digitization and automatic execution of the chemical synthesis literature. *Science*, 370:101–108, 2020. doi: 10.1126/science.abc2986.

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 (EMNLP)*, pp. 2676–2694, 2023. doi: 10.18653/v1/2023.emnlp-main.162.

Opentrons Labworks. Opentrons Python Protocol API and HTTP API reference. `https://docs.opentrons.com/`, 2026. Accessed 2026-09-06.

Gihan Panapitiya, Emily Saldanha, Heather Job, and Olivia Hess. AutoLabs: cognitive multi-agent systems with self-correction for autonomous chemical experimentation. *Scientific Reports*, 16(1): 19554, 2026. doi: 10.1038/s41598-026-45593-z.

Yusuf Roohani, Andrew Lee, Qian Huang, Jian Vora, Zachary Steinhart, Kexin Huang, Alexander Marson, Percy Liang, and Jure Leskovec. BioDiscoveryAgent: An AI agent for designing genetic perturbation experiments. In *International Conference on Learning Representations (ICLR)*, 2025.

Priyanka V. Setty, Arvind Ramanathan, Ian Foster, and Rick Stevens. Aegis: Assay-aware protocol validation and runtime monitoring for open-source liquid handling robots, 2026.

Renjian Song, Yaokai Fu, Ziyan Zhao, Jigang Yu, Qing Yuan, and Chang-Ting Chen. Towards autonomous biology: Compiler-verified protocols as a foundation for real-world AI execution. *bioRxiv*, 2026. doi: 10.64898/2026.05.05.720956. Preprint.

Nathan J. Szymanski, Bernardus Rendy, Yuxing Fei, Rishi E. Kumar, Tanjin He, David Milsted, Matthew J. McDermott, Max Gallant, Ekin Dogus Cubuk, Amil Merchant, Haegyeom Kim, Anubhav Jain, Christopher J. Bartel, Kristin Persson, Yan Zeng, and Gerbrand Ceder. An autonomous laboratory for the accelerated synthesis of inorganic materials. *Nature*, 624:86–91, 2023. doi: 10.1038/s41586-023-06734-w.

Ellis Whitehead, Fabian Rudolf, Hans-Michael Kaltenbach, and Jörg Stelling. Automated planning enables complex protocols on liquid-handling robots. *ACS Synthetic Biology*, 7(3):922–932, 2018. doi: 10.1021/acssynbio.8b00021.

Rick P. Wierenga, Stefan M. Golas, Wilson Ho, Connor W. Coley, and Kevin M. Esvelt. PyLabRobot: An open-source, hardware-agnostic interface for liquid-handling robots and accessories. *Device*, 1(4):100111, 2023. doi: 10.1016/j.device.2023.100111.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Naruki Yoshikawa, Marta Skreta, Kourosh Darvish, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Andrew Zou Li, Yuchi Zhao, Haoping Xu, Artur Kuramshin, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Large language models for chemistry robotics. *Autonomous Robots*, 47:1057–1086, 2023. doi: 10.1007/s10514-023-10136-2.

Zongzheng Zhang, Chenghao Yue, Haobo Xu, Minwen Liao, Xianglin Qi, Huan-ang Gao, Ziwei Wang, and Hao Zhao. Robochemist: Long-horizon and safety-compliant robotic chemical experimentation. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 3537–3568. PMLR, 2025.

A PLANNING SETUP AND SCORING

A.1 HARDWARE CONFIGURATION AND TASK FAMILIES

The liquid handler is an Opentrons Flex with a left single-channel 50 μL pipette and a right eight-channel 50 μL pipette, both supporting 1–50 μL per channel per stroke. Output plates are Corning 96-well flat-bottom plates with 360 μL well capacity. Flex tip racks contain 96 tips; NEST 12-channel reservoirs and conical-tube racks provide 15 mL source locations. A2 is reserved for the camera fixture and A3 for trash. Larger transfers are decomposed into strokes, and multichannel operations account for all participating channels.

The analyte is bromothymol blue (BTB), with concentrations on the suite's relative x scale. The manifest supplies 50 mL of $0.12x$ BTB, 20 mL each of $0.10x$ and $0.08x$ BTB, and 100 mL of buffer. The request name pH 10 Buffer maps to CAPS Buffer, specified as 50 mM at pH 10. BTB tasks request 0.02–$0.12x$ and 50–300 μL final volumes. These are configured inputs, not measurements of pipetting accuracy.

Table 1: BTB-48 task families. Each has four easy, four medium, and four hard requests.

Family	Requirement	Wells/task	Final μL
Normalization	Different additions or dilution of prefilled wells	3–8	50–300
Constraints	Mixtures within equipment and supply limits	8–96	90–300
Dilution	Transfers out, ratios, and retained volumes	6–16	75–100
Multi-stage	Preparation for a downstream plate	8–24	100–120

A serial-dilution instance. In `dilution_easy_10`, rows A–C and columns 1–3 form nine output wells, targeting $0.08x$, $0.04x$, and $0.02x$. Per row, loading 180 μL of $0.08x$ stock into the first point and 90 μL buffer into the other two, transferring 90 μL successively, and discarding 90 μL from the last point leaves 90 μL at each target concentration. These are total logical quantities; each transfer respects the 50 μL stroke limit after lowering.

A normalization instance. In `normalization_hard_10`, eight wells initially contain 60 μL at concentrations $(0.12, 0.09, 0.06, 0.03, 0.10, 0.08, 0.06, 0.04)x$. Buffer-only normalization to $0.03x$ requires final volumes $(240, 180, 120, 60, 200, 160, 120, 80)$ μL. The initial state and well-specific requirements must be preserved during planning.

A resource instance. In `constraint_hard_10`, 96 wells each require 240 μL at $0.04x$ from $0.12x$ stock: 80 μL stock and 160 μL buffer per well. The plate consumes 7.68 mL stock and 15.36 mL buffer. Correct per-well arithmetic must be accompanied by sufficient source capacity, allocation, and tip supply.

A.2 SCORING AND COVERAGE

All planning results use the BTB v2.2 scorer. For target $(V^\star, C^\star)$, scientific checks use

$$|V - V^\star| \le \max(0.5\ \mu\text{L}, 0.05V^\star), \qquad |C - C^\star| \le \max(0.004x, 0.12C^\star). \tag{5}$$

Every one of the 815 expected output wells has both targets. All must pass for scientific task success. Command normalization accepts documented field aliases before replay. All systems use the same frozen score, and the scorer does not repair candidate outputs.

Reference targets and scoring. BTB-48 consists of experiment-specific requests with mathematically derived per-well references. Evaluation uses these references; the agent's requirement checker independently extracts targets from the request text. Each returned plan is evaluated once, without evaluator feedback or repair.

Replay normalizes supported command aliases and executes commands in order under proportional mixing. It starts from the plan's declared initial state (empty when omitted), without inserting reference prefilled wells. Source concentrations are inferred from supported names, descriptions, and

stock strengths. Table 2 defines the reported metrics. Missing plans and generation failures count as failures; all 48 tasks remain in the denominator. Partial well correctness earns no fractional task credit. Alternative command sequences pass when they satisfy the same evaluated outcomes and bounds.

Table 2: Planning metrics. Each metric is a per-task Boolean, summed over 48 tasks.

Metric	Pass criterion
Structure	The returned command plan passes structural validation.
Science	All expected output wells meet both volume and concentration targets after replay.
Bounds	The plan satisfies the evaluated physical and resource limits.
Plan	Structure $\wedge$ Science $\wedge$ Bounds.

Replayable plans may receive a Science pass even when Structure fails; complete-plan success requires all three components.

Bounds check supported labware and pipettes, valid coordinates and slots, carried and well volumes, source supply, and total tip demand. Reservoir withdrawal by an eight-channel pipette counts all eight channels. External source loading must admit a nonnegative trajectory within container capacity. Explicit reagent bindings permit reagent-specific inventory accounting. Missing source identity is not guessed for those resource checks: such sources receive container and aggregate-inventory checks only. Scientific concentration replay retains source-name/description inference. The scores therefore do not certify every reagent identity or reagent-specific inventory independently.

The score follows command-order material replay but excludes extra protocol-order constraints, mixing effectiveness, chemical carryover, exact source paths, deadlines, and exact tip-address reuse. The scheduling family measures coupled plate preparation, not makespan optimization. Structural validity is a stored validation result rather than a complete physical execution guarantee.

Table 3: **Planning success by task family over three rounds: mean (sample standard deviation).** Each family has 12 tasks. Unrounded family means sum to the Plan mean in Table 5.

System	Norm.	Constraint	Dilution	Multi-stage
AutoBioLab / Sonnet 5	11.3 (1.2)	9.3 (1.5)	12.0 (0.0)	11.3 (1.2)
AutoBioLab / Gemini 3 Flash	11.0 (1.0)	12.0 (0.0)	9.0 (0.0)	0.7 (0.6)
AutoBioLab / DeepSeek V4 Flash	7.7 (3.1)	7.7 (0.6)	8.3 (1.5)	4.7 (3.2)
AutoBioLab / GPT-5.4-mini	0.7 (0.6)	6.7 (2.3)	0.7 (0.6)	0.0 (0.0)
Claude Code / Sonnet 5	11.7 (0.6)	9.3 (1.5)	11.7 (0.6)	8.0 (2.6)
Coscientist / Gemini 3 Flash	11.0 (0.0)	4.3 (0.6)	8.3 (0.6)	6.0 (1.0)
Opus 5	12.0 (0.0)	9.7 (0.6)	11.7 (0.6)	12.0 (0.0)
Sonnet 5	6.0 (1.0)	6.0 (2.0)	4.3 (2.3)	5.0 (1.7)
GPT-5.6 Sol	9.3 (1.2)	8.7 (0.6)	12.0 (0.0)	7.7 (0.6)
GPT-5.6 Luna	0.0 (0.0)	0.0 (0.0)	0.7 (0.6)	0.3 (0.6)
Gemini 3 Flash	8.7 (0.6)	3.3 (0.6)	6.3 (0.6)	5.3 (0.6)
GPT-5.4-mini	0.0 (0.0)	0.3 (0.6)	0.0 (0.0)	0.0 (0.0)
DeepSeek V4 Flash	3.3 (1.2)	2.3 (2.5)	2.7 (2.1)	1.7 (1.5)
Kimi K2.6	9.0 (0.0)	5.7 (1.2)	8.7 (0.6)	5.0 (0.0)

A.3 Inference settings that affect output quality

Table 4 gives per-reply limits and iterative budgets. No reasoning option was specified. CLI conditions use their own defaults, so nominal API settings do not establish their effective limits. The benchmark uses no common total-token budget or best-of-N selection.

The initial AutoBioLab/Sonnet and Claude Code collections used subscription/CLI access; later collections include API access. Effective sampling and reply limits for those initial CLI calls are unspecified. Empty or truncated model outputs count as failures. The comparisons share task inputs and scoring, but inference budgets vary across systems and rounds.

Table 4: Output-quality settings for planning and ablation. Caps are tokens per reply; retry counts are additional attempts.

Condition	Effective settings and scope
Direct Sonnet / Sol / Luna	32,768; default reasoning; temperature 0; one generation per task, without evaluation-layer repair or continuation.
Other direct API rows	32,768; temperature 0; default reasoning, one returned plan graded once.
AutoBioLab planning	16,384 requested per planning reply; temperature 0 for API calls; default reasoning in the overall collections; up to 3 schema-correction retries per step.
Scientific feedback	Up to 2 reflection iterations, with deterministic solving and keep-best acceptance when enabled.
Robotic feedback	Up to 3 patch attempts per compilation and 3 outer simulation/recompilation retries; 8,192-token patch default, with recorded 32,768-token exceptions in overall-ablation reruns. Full regeneration after an empty compilation requests 32,768.
Coscientist / Gemini	Up to 8 command-loop steps, 32,768 per model reply, temperature 0 and default reasoning.
Claude Code / Sonnet	CLI-default reasoning unless recorded otherwise; API temperature and output caps are not forwarded by the adapter.
Mechanism study	Sonnet 5, default reasoning; same planning-stage limits; robotic repair alternatives request 8,192 per reply.

A.4 THREE-ROUND PLANNING RESULTS

Table 5: **Planning benchmark over three rounds: mean (sample standard deviation).** The three repetitions each cover 48 tasks. Plan requires Structure, Science, and Bounds jointly; Plan % gives the corresponding percentage. Blocks contain AutoBioLab, agent baselines, and direct models.

System	Plan /48	Plan %	Structure	Science	Bounds
AutoBioLab / Sonnet 5	44.0 (1.0)	91.7 (2.1)	47.3 (0.6)	44.3 (0.6)	47.0 (1.0)
AutoBioLab / Gemini 3 Flash	32.7 (0.6)	68.1 (1.2)	45.0 (2.0)	33.3 (0.6)	40.7 (2.5)
AutoBioLab / DeepSeek V4 Flash	28.3 (6.4)	59.0 (13.2)	41.3 (2.3)	30.3 (4.6)	39.3 (2.5)
AutoBioLab / GPT-5.4-mini	8.0 (1.7)	16.7 (3.6)	41.3 (2.3)	8.3 (2.1)	38.3 (2.3)
Claude Code / Sonnet 5	40.7 (4.0)	84.7 (8.4)	45.7 (1.5)	44.3 (2.1)	44.0 (2.6)
Coscientist / Gemini 3 Flash	29.7 (1.5)	61.8 (3.2)	47.0 (0.0)	34.0 (1.0)	39.3 (1.5)
Opus 5	**45.3 (1.2)**	**94.4 (2.4)**	47.7 (0.6)	47.7 (0.6)	45.3 (1.2)
Sonnet 5	21.3 (2.3)	44.4 (4.8)	25.3 (4.2)	40.3 (1.5)	23.0 (2.0)
GPT-5.6 Sol	37.7 (1.5)	78.5 (3.2)	48.0 (0.0)	39.3 (2.1)	46.0 (0.0)
GPT-5.6 Luna	1.0 (1.0)	2.1 (2.1)	29.7 (2.5)	4.7 (3.2)	11.7 (0.6)
Gemini 3 Flash	23.7 (0.6)	49.3 (1.2)	48.0 (0.0)	28.3 (1.2)	36.3 (0.6)
GPT-5.4-mini	0.3 (0.6)	0.7 (1.2)	18.7 (2.5)	2.7 (2.5)	18.0 (1.7)
DeepSeek V4 Flash	10.0 (2.6)	20.8 (5.5)	32.3 (1.5)	25.0 (1.7)	15.3 (3.2)
Kimi K2.6	28.3 (1.5)	59.0 (3.2)	34.3 (3.8)	33.0 (3.5)	31.0 (1.0)

For each system, Round 1 is the initial benchmark collection and Rounds 2–3 are reruns s101 and s102. Each round includes all 48 tasks, including model failures; the identifiers label replicate collections rather than provider-controlled random seeds. For a per-round score x_r, the reported mean and sample standard deviation are

$$\bar{x} = \frac{1}{3}\sum_{r=1}^{3} x_r, \qquad s = \sqrt{\frac{1}{2}\sum_{r=1}^{3}(x_r - \bar{x})^2}. \tag{6}$$

The same three rounds contribute to every metric, including token use. We aggregate the rounds as collected, so the observed spread includes acquisition-setting differences as well as sampling variation; it is not a confidence interval or a measure of uncertainty over unseen tasks. Within a round, token means use tasks with recorded usage; all tasks remain in the success denominator. Provider token accounting differs, making token totals descriptive rather than matched compute. Values are rounded to one decimal place only after aggregation.

Table 6: **Individual planning rounds and reported token use.** Plan counts are out of 48; tokens are thousands per task, reported as mean (sample standard deviation) over the same three rounds. These counts underlie Table 5.

System	Round 1	Round 2	Round 3	ktok/task
AutoBioLab / Sonnet 5	45	44	43	167.1 (184.8)
AutoBioLab / Gemini 3 Flash	32	33	33	82.3 (5.5)
AutoBioLab / DeepSeek V4 Flash	21	32	32	92.7 (30.7)
AutoBioLab / GPT-5.4-mini	9	9	6	70.5 (7.8)
Claude Code / Sonnet 5	45	37	40	262.7 (33.6)
Coscientist / Gemini 3 Flash	30	28	31	17.4 (0.7)
Opus 5	44	46	46	25.4 (23.6)
Sonnet 5	20	24	20	16.5 (0.4)
GPT-5.6 Sol	38	39	36	10.7 (0.3)
GPT-5.6 Luna	0	1	2	8.7 (0.6)
Gemini 3 Flash	23	24	24	7.2 (0.2)
GPT-5.4-mini	0	0	1	8.9 (1.0)
DeepSeek V4 Flash	13	8	9	15.7 (0.4)
Kimi K2.6	27	30	28	21.3 (1.6)

A.5 THREE-ROUND OVERALL ABLATION

M1–M4 use Sonnet 5 with the scientific and robotic groups independently enabled. Each variant is evaluated over three rounds on the same 48 requests. Tables 7 and 8 report arithmetic means and sample standard deviations across those three round-level scores. We neither select the best result for each task nor combine repeated outcomes by majority vote. Benchmark AutoBioLab and factorial M4 use their respective experiment collections.

Table 7: **Overall M1–M4 ablation over three rounds: mean (sample standard deviation).** Counts are out of 48. S enables quantity synthesis, scientific checks, and design repair; R enables command validation and repair. All metrics use the same three rounds.

Variant	S	R	Plan /48	Structure	Science	Bounds
M1	off	off	22.3 (9.0)	26.3 (10.8)	31.3 (1.5)	23.3 (9.9)
M2	off	on	29.0 (7.0)	31.7 (4.9)	36.7 (6.4)	30.7 (6.7)
M3	on	off	42.3 (2.1)	47.0 (1.7)	47.0 (1.7)	42.7 (1.5)
M4	on	on	44.7 (0.6)	47.3 (0.6)	46.0 (1.7)	46.0 (1.7)

Table 8: **Individual ablation rounds and cost.** Plan counts are out of 48. Percentages and thousands of reported tokens per task are three-round mean (sample standard deviation).

Variant	Round 1	Round 2	Round 3	Plan %	ktok/task
M1	12	28	27	46.5 (18.7)	93.7 (86.9)
M2	21	34	32	60.4 (14.6)	203.1 (218.2)
M3	43	44	40	88.2 (4.3)	100.2 (91.8)
M4	44	45	45	93.1 (1.2)	153.9 (151.5)

Compiler handling of transfer limits changed after Round 1. In Round 2, M2 uses a 32,768-token repair cap and M4 uses 8,192. In Round 3, M2 uses 32,768 for the first 36 tasks and 8,192 for the remaining 12; M4 uses 32,768 throughout. M1 and M3 do not invoke robotic repair. The reported spread therefore includes compiler and budget differences as well as sampling variation.

The contrasts describe the configured module groups on the fixed task set. Solver-delegating prompts may leave quantities unspecified when S is disabled. The paired reply replay below isolates solver effects while holding model replies fixed.

Table 9: **Within-round changes in complete-plan success.** Entries count additional tasks passed by the destination variant. The final column gives the mean and sample SD of these three differences, rather than pooling task-runs as independent samples.

Comparison	Round 1	Round 2	Round 3	Mean (SD)
M1 → M3	31	16	13	20.0 (9.6)
M2 → M4	23	11	13	15.7 (6.4)
M1 → M2	9	6	5	6.7 (2.1)
M3 → M4	1	1	5	2.3 (2.3)

B MECHANISM ABLATION DETAILS

B.1 CHAIN TASKS AND PAIRED SOLVER REPLAY

The mechanism study uses 12 additional serial-dilution tasks and Sonnet 5 with default reasoning. Requests specify concentrations, retained volumes, and layout while leaving transfer quantities to the planner. Table 10 gives the task parameters. For dilution factor f and retained volume V, one reference construction uses transfer $T = V/(f-1)$, buffer prefill V, head loading $V+T$, and final discard T. References use exact rational arithmetic, whole-microlitre quantities, at most 360 μL per well, and minimum concentration $0.0125x$. Scoring accepts other quantity schemes reaching the same targets; it does not separately certify the transfer route or mixing procedure.

Intermediate plans are compiled with scientific and robotic repair disabled, then scored using the same Structure, Science, and Bounds criteria as BTB-48. Uncompilable plans fail. In Figure 3a, four sets of model replies are replayed unchanged under no solver, local solving, and joint solving. Two sets were collected without solvers and two with synthesis only; solver changes occur after the last model reply. Table 11 gives all per-run counts. Joint solving corrects 27 failures without losses across the 48 task-runs.

Table 10: Chain-task parameters. ID is the numeric task suffix; f is the dilution factor and C_0 the stock concentration. Point i targets C_0/f^{i-1} and retained volume V in every listed row.

ID	Points	f	C_0 (x)	V (μL)	Rows	Columns
50	2	4	0.12	90	C–D	2, 5
51	2	5/3	0.10	60	A–C	7, 8
52	2	4/3	0.08	50	G–H	10, 12
53	3	2	0.08	70	D–F	4, 5, 6
54	3	5/3	0.12	80	A–B	1, 5, 9
55	3	3	0.12	60	E–H	8, 10, 12
56	4	4/3	0.12	60	F–G	1, 3, 5, 7
57	4	5/3	0.10	90	B–D	6, 7, 8, 9
58	5	4/3	0.10	50	A–B	3, 5, 7, 9, 11
59	5	3/2	0.10	80	C–E	8, 9, 10, 11, 12
60	6	4/3	0.08	40	E–F	2, 4, 6, 8, 10, 12
61	6	5/4	0.10	70	G–H	1, 2, 3, 4, 5, 6

Table 11: Complete-plan successes for Figure 3a,c. Each run contains 12 tasks. Runs are paired within each panel; panels use separate initial samples.

Panel	Condition	Run 1	Run 2	Run 3	Run 4	Mean (SD)
a	no solvers	1	4	4	1	2.50 (1.73)
a	+ local solver	1	4	4	1	2.50 (1.73)
a	+ joint solver	11	10	9	7	9.25 (1.71)
c	first plan	6	8	8	9	7.75 (1.26)
c	+ re-solve	9	8	10	9	9.00 (0.82)
c	+ rewrite	10	10	11	9	10.00 (0.82)

Table 12: **Checker effects on complete-plan success.** Three fresh, paired repetitions of the same 12 chain tasks. Each initial plan is shared by all six conditions; means use sample SD.

Solver	Checks	Run 1	Run 2	Run 3	Mean (SD)	Success (%)
Off	No check	3	1	3	2.33 (1.15)	19.4
Off	Basic check	4	2	5	3.67 (1.53)	30.6
Off	Basic + requirement	5	2	5	4.00 (1.73)	33.3
On	No check	10	9	9	9.33 (0.58)	77.8
On	Basic check	11	11	9	10.33 (1.15)	86.1
On	Basic + requirement	12	11	11	11.33 (0.58)	94.4

B.2 Scientific repair traces

Figure 3c follows four stored 12-task reply sets through target-informed re-solving and requirement-aware rewriting. These samples differ from the checker experiment below. Re-solving makes no model calls; rewriting uses eight calls for three additional repairs in the first two runs and six calls for one additional repair in the other two. The traces include partial repair reruns after changes to target feedback and acceptance scoring. They describe corrections to stored plans, rather than four repetitions of a fixed end-to-end system.

B.3 Paired checker experiment

We generate 36 fresh initial plans (12 tasks $\times$ three repetitions) with Sonnet 5, temperature 0 and reasoning disabled. Each pre-solver plan is cloned into all six conditions: solvers off/on $\times$ no check/basic check/basic plus requirement check. Repetition IDs are not provider seeds. Basic checks cover destination wells, volume balance, concentration balance, outcome self-consistency, and stock/recipe consistency. Requirement checking additionally compares net final volumes and declared outcomes with independently extracted task targets. Both checked conditions share one target extraction and receive the same target block when repairing; only the active checks trigger feedback and determine acceptance.

After a trigger, the common policy attempts target-informed re-solving when solvers are enabled, then at most two model rewrites with Pareto acceptance and early stopping after rejection. No check permits no scientific repair. Re-solving and post-rewrite solving are disabled throughout solver-off conditions. Planning/rewrite replies allow 16,384 tokens, target extraction 2,048; schema-validated planning stages allow three extra retries. Robotic repair is disabled. Actual calls can differ because checks trigger different repairs; the maximum repair policy is fixed, not total inference cost. Final plans use identical deterministic compilation and the joint task scorer.

Relative to no checking, basic/full checking adds four/five successful task-runs without solvers and three/six with solvers, with no losses. Relative to basic checking, requirement checking yields two gains and one loss without solvers, and three gains without losses with solvers. The latter are task 52 in repetitions 1 and 3 and task 58 in repetition 3: all escape the basic gate and pass after requirement-triggered deterministic re-solving, with no repair-model call. The solver-off loss is a different rewrite outcome on task 56; its initial plan already fails.

Repair calls across the 36 task-runs total 0/27/34 without solvers and 0/6/8 with solvers for no/basic/requirement checking. Each condition shares 144 initial planning calls; checked conditions additionally share 36 target-extraction calls.

B.4 Robotic repair

Four repair policies act on the same nine robotically invalid plans in three repetitions, using an 8,192-token reply cap. All nine failures are detected. Localized patching edits diagnosed commands; blind retry hides inner-loop diagnostics; whole-plan rewriting regenerates all commands from the same diagnostics; patching without the outer loop omits subsequent simulation and recompilation. Table 13 gives the data for Figure 3d.

Table 13: **Repeated robotic repair on the same nine invalid plans.** Entries separated by a slash give repetitions 1 and 2; calls are totals per repetition. The final column averages calls divided by repaired plans within each repetition, matching Figure 3d. Every mode uses an 8,192-token per-reply cap, without equalizing total calls.

Repair mode	Repaired (of 9)	Calls	Calls/repair
Localized patch	8 / 7	21 / 15	2.4
Inner-loop blind retry	6 / 6	67 / 66	11.1
Whole-plan rewrite	4 / 6	23 / 21	4.6
Localized patch, no outer loop	6 / 8	14 / 14	2.0

Blind retry retains outer-loop feedback: six of nine cases receive full diagnostics through that loop in the first repetition. Whole-plan rewriting can truncate at the reply cap. The outer loop repairs two more cases in the first repetition and one fewer in the second, providing no clear additional benefit. These two repetitions support descriptive comparisons of repair success and cost.

No mode damages robotic validity on the 39 initially valid plans. In the first repetition, complete-plan success rises from 21/48 to 23, 24, 21, and 23 for localized patching, blind retry, whole-plan rewriting, and patching without the outer loop, respectively. Robotic correction can leave scientific errors unresolved.

B.5 Additional detection results

On 48 stored plans, basic scientific checks flag 18 of 27 independently scored failures; adding task-requirement checks flags 25. Neither condition flags any of the 21 correct plans. The two remaining failures involve reservoir addresses and are detected by robotic checks. This measures detection, separately from the paired task-success experiment.

Robotic detection is evaluated on 72 chain plans and 191 BTB-48 plans. Table 14 reports detection and stroke-splitting results. Detection and final bounds scoring share physical rules; these results do not constitute physical measurements.

Table 14: Additional robotic detection and stroke-splitting results. Counts of detected errors use the invalid-plan denominator; complete-plan passes use all plans.

Metric	Chain (72 plans)	BTB-48 (191 plans)
Invalid plans	9	48
Detected by structural validation	3/9	33/48
Detected by dry run	3/9	33/48
Detected by bounds checks	9/9	48/48
False alarms on valid plans	0/63	0/143
Complete-plan passes, stroke splitting on	40/72	139/191
Complete-plan passes, stroke splitting off	0/72	3/191

B.6 Case-study data

Figure 4a–d uses chain tasks 58, 54, 61, and 53, respectively (Table 10). Tables 15 and 16 list the displayed volumes and repair traces. In panel c, the initial design fills A1–A6 and leaves all 12 required wells G1–G6 and H1–H6 empty. The requirement-aware rewrite fills those 12 wells and passes the joint scorer; each targets 70 μL at concentrations $0.10(4/5)^{i-1}x$, $i = 1, \ldots, 6$.

In panel d, invalid reservoir addresses cause 18 missing-well errors and 18 empty-tip errors. The shown aspiration requests 35 μL. Localized patching changes addresses `1/2` to `A1/A2` without changing the recipe.

Table 15: Volume data for Figure 4a,b, in μL. Vectors follow the chain order; transfers list adjacent-well moves. The panel-a joint solution discards 150 from the last point; the panel-b corrected solution uses no final discard.

Panel	Plan	Head load	Buffer prefills	Transfers	Final volumes
a	Initial	50	15, 15, 15, 15	45, 45, 45, 45	5, 15, 15, 15, 50
a	Joint solver	200	50, 50, 50, 50	150, 150, 150, 150	50, 50, 50, 50, 50
b	Initial	80	32, 32	48, 48	32, 32, 80
b	Re-solved	156.8	51.2, 32	76.8, 48	80, 80, 80

Table 16: Repair trace data for Figure 4d. Each list gives remaining robotic issues after successive model calls. The initial plan has 36 issues.

Method	Issues after calls	Calls with diagnostics	Final plan
Localized patch	0	1	Pass
Blind retry	36, 36, 30, 0	4	Pass
Whole-plan rewrite	0, 0, 0	1, 2, 3	Incorrect volumes

C Runtime recovery protocol

C.1 Evaluation setting and scenario sampling

Recovery uses the system's session executor, orchestration, and policies with simulated robot and camera endpoints. A separate world model tracks liquids and tips for scoring. Faults can cause divergence between agent-visible state and actual simulated state. All policies share the executor, environment, observation interfaces, and available actions.

Each of three rounds contains 96 scenarios on 14 base plans: 30 single faults, 27 resource faults, and 39 persistent or compound faults. Outcomes are labeled as recovery (83), abort (3), or escalation (10). The rounds use scenario seeds 101 and 102, preserving the task and fault catalogue while redrawing fault positions. Every policy is evaluated on every scenario. Means and sample standard deviations use the two round-level scores; the spread does not estimate uncertainty over unseen tasks.

The 96 scenarios retain approximately half of each source-suite/fault-variant group from a 191-scenario design, alternating rounding for odd group sizes and prioritizing less-represented tasks. All base tasks and fault variants remain represented. The 191-scenario collection is excluded from the reported averages.

Base plans and preparation. The 14 AutoBioLab/Sonnet plans span normalization, constraints, dilution, and multi-stage preparation. Selection requires structural validation, rather than a perfect independent Science score. Visual checkpoints expect the preceding stroke volume after aspiration and zero after dispense or mixing. Resource variants add compatible spare racks or reservoirs with matching source liquid; both fault-free and faulted executions use the same prepared plan. Spare-resource variants produce 24 prepared cases from the 14 base plans. Outcome labels account for all compatible supply already on deck. Faults change execution or observations while leaving scientific targets fixed.

C.2 Policies and scoring

Stop on error aborts at the first detected error; naive retry homes and retries the interrupted command. Our recovery agent uses a ReAct tool loop over session state, camera observation, verified finish, bounded corrective actions, labware substitution, resumption, and handover, without deterministic recovery recipes. Raw policies select the same recovery actions through a chat interface without the runtime's state tracking, belief correction, or verified finish. These are system comparisons; equal action budgets do not imply equal prompts, reasoning settings, or numbers of model calls.

For a recoverable scenario, success requires completion and agreement with its fault-free reference within 1 μL per evaluated well and 0.02 absolute liquid-fraction difference. Source-supply and spare labware are excluded from this output comparison. For a stop-required scenario, correct handling requires stopping or escalating without completion or recorded unsafe actions. We report recovery over the 83 recoverable cases, handover over the 13 stop-required cases, and correct handling over all 96 cases. The denominator is applied within each round before averaging.

World-recorded unsafe actions per 100 episodes count events, not episodes: repeated violations can exceed 100 per 100 episodes. Correct handling without an unsafe action instead counts episodes satisfying both conditions. Dry aspiration is recorded separately as waste. Calls and reported tokens are averaged per episode within each round, then summarized over the three rounds. A fault-free reference can itself contain a planning error, so matching it is distinct from passing all independent scientific targets.

Outcome judgment. Each prepared plan is first executed without faults to obtain its reference state. The faulted outcome is compared over the union of evaluated wells, excluding spare labware and supply labware identified by seeded source volumes above 1,000 μL. Recovery requires completion and reference agreement. Abort-labeled cases permit abort or escalation; escalation-labeled cases require escalation. Both stop-required labels require no completion and no recorded unsafe action. For each round,

$$\text{Correct} = \frac{N_{\text{recovered}} + N_{\text{handed over}}}{96}, \quad \text{Recover} = \frac{N_{\text{recovered}}}{83}, \quad \text{HandOver} = \frac{N_{\text{handed over}}}{13}. \tag{7}$$

Rates are reported as percentages. Unsafe/100 is 100 times the total unsafe-action count divided by 96; the unsafe-episode percentage counts episodes with at least one recorded unsafe action, divided by 96; safe-correct counts correctly handled episodes with zero recorded unsafe actions.

Inference and observation settings. Each episode permits at most three recovery rounds, including compound faults. Both model-policy types allow 15 turns and ten corrective actions per round. AutoBioLab uses an 8,192-token reply cap; raw policies use 4,096 and receive the next six protocol commands plus the accumulated recovery transcript. API calls use temperature 0 and default reasoning, except raw Kimi uses reasoning disabled. The simulated vision tolerance is 5 μL, while final grading uses 1 μL. Thus visually undetected residuals may fail the final score. No physical plate measurements or real camera images enter these results.

C.3 Fault construction and scope

Six base plans were used during recovery development and eight introduced afterwards. Results pool these plans, with breakdowns by fault kind.

Single faults cover delivery errors, droplets, stalls, tip errors, observation failures, and emergency stops. Resource faults vary remaining tips or source liquid and the availability of replacements; stop/recover labels depend on all compatible supply already on deck. Compound cases combine missing tips with partial aspiration, overpressure with droplets, lost pickup acknowledgement with retained dispense, unknown errors with stalls, depletion with false alarms, or exhausted tips with another missing tip. A policy may fail before a later scheduled fault is reached; eligibility requires a fault to fire, not every scheduled event to fire.

A scripted oracle with access to world state checks labeled feasibility; model policies cannot access that state. Scenarios sharing base plans are correlated, so the 288 episodes are not treated as independent task samples for significance testing.

D Complete recovery results

D.1 Recovery under changed execution conditions

We apply every policy to all three 96-scenario rounds and score the resulting simulator world state. A case is handled correctly when it either completes with outputs matching the fault-free reference

or stops appropriately when recovery is infeasible. Table 17 summarizes the three round-level rates; Table 18 gives their counts and token use.

Table 17: **Runtime recovery on the compact suite: mean (sample standard deviation) over three rounds.** Each round has 96 scenarios: 83 recoverable and 13 stop-required. Recover and Hand-over use these conditional denominators; Correct uses all 96. Unsafe /100 counts world-recorded unsafe actions per 100 episodes and can exceed 100; Unsafe ep. % is the percentage of episodes with at least one such action, as plotted in Figure 2b. Calls is model calls per episode. Round 1 is the initial collection; Rounds 2–3 are s101 and s102.

Policy	Recover %	Correct %	Hand-over %	Unsafe /100	Unsafe ep. %	Calls
Stop on error	0.0 (0.0)	13.5 (0.0)	100.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
Naive retry	19.3 (0.0)	16.7 (0.0)	0.0 (0.0)	18.1 (4.8)	4.9 (0.6)	0.0 (0.0)
Ours / Sonnet 5	79.5 (3.2)	82.3 (2.8)	100.0 (0.0)	5.6 (1.6)	4.2 (0.0)	8.8 (0.3)
Ours / Gemini 3 Flash	79.1 (1.8)	81.6 (2.2)	97.4 (4.4)	10.4 (2.1)	4.2 (0.0)	10.8 (0.4)
Ours / DeepSeek V4 Flash	75.5 (6.6)	78.8 (5.7)	100.0 (0.0)	10.8 (0.6)	4.5 (0.6)	10.2 (0.0)
Ours / GPT-5.4-mini	39.8 (2.1)	47.9 (1.8)	100.0 (0.0)	15.3 (8.0)	8.3 (3.1)	7.2 (0.3)
Raw / Sonnet 5	63.9 (3.2)	66.3 (3.2)	82.1 (4.4)	90.3 (8.9)	26.7 (2.6)	2.8 (0.1)
Raw / Gemini 3 Flash	49.4 (2.1)	53.1 (2.1)	76.9 (7.7)	10.1 (1.2)	6.6 (0.6)	2.8 (0.1)
Raw / DeepSeek V4 Flash	56.2 (4.2)	58.7 (2.6)	74.4 (8.9)	74.3 (12.6)	29.9 (1.6)	3.7 (0.2)
Raw / GPT-5.4-mini	30.9 (3.5)	37.2 (2.6)	76.9 (13.3)	54.5 (8.1)	28.8 (2.6)	3.2 (0.1)
Raw / Opus 5	65.5 (5.6)	68.1 (4.9)	84.6 (7.7)	114.9 (12.9)	30.6 (3.2)	3.1 (0.1)
Raw / GPT-5.6 Sol	63.5 (3.9)	66.7 (1.8)	87.2 (11.8)	73.3 (24.7)	27.4 (4.2)	2.8 (0.1)
Raw / GPT-5.6 Luna	46.2 (3.0)	51.4 (3.3)	84.6 (13.3)	60.4 (10.3)	26.4 (4.2)	2.8 (0.1)
Raw / Kimi K2.6	55.8 (4.6)	60.1 (4.2)	87.2 (4.4)	38.2 (9.4)	16.7 (2.8)	4.1 (0.2)

AutoBioLab improves correct handling for all four paired backbones in every round (Table 19), using 7.2–10.8 calls per episode versus 2.8–3.7 for the corresponding raw policies. With Sonnet, DeepSeek, and GPT-5.4-mini, AutoBioLab correctly handles every stop-required case in all three rounds; with Gemini it misses one of 13 in Round 1. Sonnet's unsafe-action rate decreases from 90.3 to 5.6 per 100 episodes, while Gemini's stays near its raw level (10.4 versus 10.1). Figure 5 distinguishes correct outcomes from correct outcomes without recorded unsafe actions.

Table 18: **Individual compact recovery rounds.** Round 1 is the initial collection; Rounds 2–3 are reruns s101 and s102. Correct counts are mean (sample standard deviation) over the three rounds; thousands of tokens per episode use Rounds 2–3 only, because Round 1 did not log token usage.

Policy	Round 1 /96	Round 2 /96	Round 3 /96	Correct /96	ktok/episode
Stop on error	13	13	13	13.0 (0.0)	0.0 (0.0)
Naive retry	16	16	16	16.0 (0.0)	0.0 (0.0)
Ours / Sonnet 5	80	76	81	79.0 (2.6)	124.0 (6.2)
Ours / Gemini 3 Flash	76	79	80	78.3 (2.1)	96.2 (3.9)
Ours / DeepSeek V4 Flash	73	82	72	75.7 (5.5)	106.2 (0.9)
Ours / GPT-5.4-mini	44	47	47	46.0 (1.7)	73.7 (1.2)
Raw / Sonnet 5	67	63	61	63.7 (3.1)	17.0 (0.4)
Raw / Gemini 3 Flash	49	51	53	51.0 (2.0)	9.2 (0.1)
Raw / DeepSeek V4 Flash	59	54	56	56.3 (2.5)	16.5 (1.3)
Raw / GPT-5.4-mini	38	36	33	35.7 (2.5)	10.0 (0.2)
Raw / Opus 5	67	60	69	65.3 (4.7)	15.7 (0.6)
Raw / GPT-5.6 Sol	65	62	65	64.0 (1.7)	9.1 (0.4)
Raw / GPT-5.6 Luna	47	48	53	49.3 (3.2)	9.7 (0.1)
Raw / Kimi K2.6	60	53	60	57.7 (4.0)	13.8 (0.6)

D.2 Fault-kind results and remaining failures

Table 20 uses the same three rounds to separate single faults, resource changes, and persistent or compound faults. With Sonnet, mean correct handling declines from 98.9% on single faults to 77.8% on resource changes and 72.6% on persistent or compound faults. GPT-5.4-mini remains weaker on resource changes (25.9%) and persistent or compound faults (31.6%), even though its overall same-backbone gain is positive.

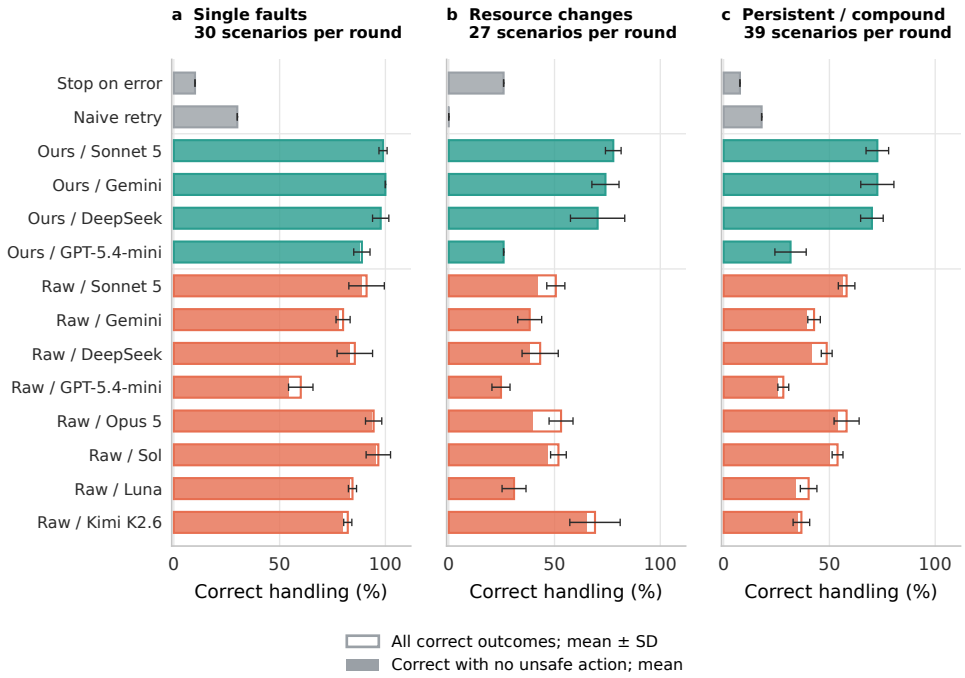

Figure 5: **Recovery outcomes by fault kind over three compact rounds.** Outlined bars show mean correct handling; filled bars retain correct episodes with no recorded unsafe action. Whiskers give one sample standard deviation of correct handling across the three rounds. Denominators are 30, 27, and 39 scenarios per round. These are pooled fault-kind groups, not held-out suites.

Table 19: **Same-backbone recovery comparisons over three rounds.** Values are mean (sample standard deviation). Gain is computed within each round before averaging, in percentage points. The three rounds reuse base plans and do not establish uncertainty over unseen tasks.

Backbone	Raw correct %	Ours correct %	Gain (points)
Sonnet 5	66.3 (3.2)	82.3 (2.8)	16.0 (4.2)
Gemini 3 Flash	53.1 (2.1)	81.6 (2.2)	28.5 (0.6)
DeepSeek V4 Flash	58.7 (2.6)	78.8 (5.7)	20.1 (7.9)
GPT-5.4-mini	37.2 (2.6)	47.9 (1.8)	10.8 (4.2)

The lost-pickup-acknowledgement plus retained-dispense pairing remains a shared failure: all four agent backbones score 0/4 in each round. Their recorded unsafe actions on these cases reflect attempts to pick up a tip when one is already attached. On the unknown-error plus two-home-stall pairing, Sonnet handles 1/4, 0/4, and 1/4, showing that the verification-and-handover policy still misses recoverable cases. Each variant contains four scenarios per round in the compact selection.

Table 20: **Correct handling by fault kind: mean (sample standard deviation) counts across three rounds.** Each row uses the same 30 single-fault, 27 resource, and 39 persistent/compound scenarios per round. Unrounded group means sum to the mean correct count in Table 18.

Policy	Single /30	Resource /27	Persistent or compound /39
Stop on error	3.0 (0.0)	7.0 (0.0)	3.0 (0.0)
Naive retry	9.0 (0.0)	0.0 (0.0)	7.0 (0.0)
Ours / Sonnet 5	29.7 (0.6)	21.0 (1.0)	28.3 (2.1)
Ours / Gemini 3 Flash	30.0 (0.0)	20.0 (1.7)	28.3 (3.1)
Ours / DeepSeek V4 Flash	29.3 (1.2)	19.0 (3.5)	27.3 (2.1)
Ours / GPT-5.4-mini	26.7 (1.2)	7.0 (0.0)	12.3 (2.9)
Raw / Sonnet 5	27.3 (2.5)	13.7 (1.2)	22.7 (1.5)
Raw / Gemini 3 Flash	24.0 (1.0)	10.3 (1.5)	16.7 (1.2)
Raw / DeepSeek V4 Flash	25.7 (2.5)	11.7 (2.3)	19.0 (1.0)
Raw / GPT-5.4-mini	18.0 (1.7)	6.7 (1.2)	11.0 (1.0)
Raw / Opus 5	28.3 (1.2)	14.3 (1.5)	22.7 (2.3)
Raw / GPT-5.6 Sol	29.0 (1.7)	14.0 (1.0)	21.0 (1.0)
Raw / GPT-5.6 Luna	25.3 (0.6)	8.3 (1.5)	15.7 (1.5)
Raw / Kimi K2.6	24.7 (0.6)	18.7 (3.2)	14.3 (1.5)

Table 21: Correct handling with no recorded unsafe action, for the filled bars in Figure 5. Entries are counts for Rounds 1 / 2 / 3. Divide by the column denominator and average to obtain the plotted percentage.

Policy	Single /30	Resource /27	Persistent or compound /39
Stop on error	3 / 3 / 3	7 / 7 / 7	3 / 3 / 3
Naive retry	9 / 9 / 9	0 / 0 / 0	7 / 7 / 7
Ours / Sonnet 5	30 / 29 / 30	20 / 21 / 22	30 / 26 / 29
Ours / Gemini 3 Flash	30 / 30 / 30	21 / 18 / 21	25 / 31 / 29
Ours / DeepSeek V4 Flash	28 / 30 / 30	17 / 23 / 17	28 / 29 / 25
Ours / GPT-5.4-mini	28 / 26 / 25	7 / 7 / 7	9 / 14 / 14
Raw / Sonnet 5	29 / 24 / 27	11 / 13 / 10	23 / 22 / 21
Raw / Gemini 3 Flash	23 / 23 / 24	10 / 9 / 12	14 / 17 / 15
Raw / DeepSeek V4 Flash	27 / 25 / 23	10 / 8 / 13	14 / 18 / 17
Raw / GPT-5.4-mini	19 / 15 / 15	6 / 8 / 6	10 / 11 / 9
Raw / Opus 5	28 / 27 / 29	10 / 10 / 12	21 / 19 / 23
Raw / GPT-5.6 Sol	29 / 27 / 30	11 / 14 / 13	20 / 19 / 20
Raw / GPT-5.6 Luna	24 / 25 / 26	8 / 7 / 10	13 / 14 / 13
Raw / Kimi K2.6	24 / 24 / 24	19 / 13 / 21	15 / 13 / 13