CHEMHARNESS: A SELF-EVOLVING SYSTEM FOR CHEMICAL REASONING VIA ASYMMETRIC DUAL-STREAM VERIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving harnesses improve a language model by consolidating its verified experience into reusable memory, but real-world chemical reasoning problems lack ground-truth answers for verification, and repeated samples of one model tend to repeat the same errors. We present ChemHarness, a closed-loop self-evolving harness for chemical reasoning driven by dual-stream self-verification. ChemHarness solves each problem through two streams of one language model: a neural stream derives the answer in natural language, and a symbolic stream writes a program that an interpreter executes. Over repeated rounds, cross-stream verification compares the two answers and turns them into label-free feedback in the form of cross-verified answers, intermediate lemmas, and discrepancies. ChemHarness organizes this feedback through a procedural tool library across problems, an episodic short-term memory within a problem, and an asymmetric information flow that withholds both memories from the neural stream, which therefore remains an independent verifier. On the chemistry datasets of SciBench, ChemHarness improves direct reasoning by 6.4 to 12.7 points on Qwen2.5-7B, 14B, 32B, and DeepSeek-V4.1-Flash, and exceeds ChemAgent by 23.3 and 22.9 points on Qwen2.5-7B and 14B. We further find that cross-stream verification is markedly more precise than single-stream consistency, with a precision of 63.0% to 75.4% against 46.2% to 55.7%, and that memory reaching the neural stream lowers the precision. Dual-stream self-verification thus allows a harness to evolve in chemical reasoning without ground-truth labels, parameter updates, or external supervision.

1 INTRODUCTION

Chemical reasoning requires a single solution to be correct in concepts and in numbers at once. A college-level chemistry problem asks the solver to select the governing law, to recognize the conditions that the problem leaves implicit, such as a reversible or an adiabatic process, to convert units and substitute physical constants, and to carry out a multi-step numerical computation (Wang et al., 2023b; Ouyang et al., 2023). Errors of language models arise at each of these steps, and a wrong constant or a wrong unit conversion suffices to invalidate an otherwise sound solution (Tang et al., 2025). Under the skill taxonomy of SciBench, 72% of the errors of direct reasoning on four backbones lie in the identification of assumptions or in calculation, two skills that enter a solution at different stages. Chemical reasoning therefore calls for systems that improve from their own errors, and such systems depend on a signal that chemistry does not supply.

A self-evolving harness improves only as far as its feedback separates correct experience from incorrect experience. Existing self-evolving frameworks admit new skills, tools, or memory by relying on environmental states (Wang et al., 2023a), unit tests on labeled examples (Cai et al., 2024), or a judge that inspects web pages and test results (Ouyang et al., 2026), and the quality of the adaptation depends critically on the quality of the feedback (Zhang et al., 2026). In chemistry, ChemAgent builds its library from the reference solutions of a development set and admits a new solution only when the solution is correct (Tang et al., 2025). An unsolved chemistry problem offers none of these signals. The problem asks for a single number, and a program that executes without error can still apply a wrong constant or a wrong condition. The remaining signal is the self-judgment

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.

of the model, which is unreliable. Language models cannot reliably estimate their own limitations in chemistry (Mirza et al., 2025), and without external feedback, self-correction fails (Huang et al., 2024). Consistency among samples fares little better. When two solutions sampled from the same model with the same prompt reach the same value, the value is correct in only 46.2% to 55.7% of cases across four backbones, because the errors of samples from one model are correlated (Kim et al., 2025). Self-evolution in chemistry therefore needs feedback that is internal to the model and yet independent of any single line of reasoning.

Our key insight is that the complementary nature of chemical reasoning, which combines conceptual derivation with symbolic computation, provides this missing independent verification signal. The same model can solve a problem via two distinct computational mechanisms: a neural stream, which derives the solution in natural language and calculates in text, and a symbolic stream, which formulates a program executed by an external interpreter. These two mechanisms exhibit orthogonal failure modes: arithmetic slips in text do not survive execution, while procedural or interface errors in code do not appear in linguistic derivations (Song et al., 2025). When both streams independently arrive at an identical numerical value and are confirmed by the model acting as an internal semantic judge, that value is rarely wrong (achieving 63.0% to 75.4% precision across four backbones, as reported in our experiments). We term this check dual-stream self-verification. This verification also operates below the final outcome: intermediate quantities that both streams establish yield verified lemmas, whereas variables diverging across iterative attempts denote discrepancies, providing rich, label-free feedback even in rounds where final answers differ.

We instantiate this insight in ChemHarness, a closed-loop self-evolving harness driven by two tightly coupled components: dual-stream self-verification and feedback extraction and memory evolution and asymmetric information flow. In every round, ChemHarness concurrently derives a natural-language solution and executes a program, transforming cross-stream verification into label-free feedback in the form of cross-verified answers, intermediate lemmas, and discrepancies. Verified experience is then consolidated across two temporal scopes: verified solutions are abstracted into a procedural tool library via self-constructed unit tests, while unverified rounds accumulate lemmas and discrepancies in an episodic short-term memory to ground a final fallback solution. Crucially, ChemHarness strictly confines these memories to the symbolic stream and the fallback solver while leaving the neural stream unconditioned. This verifier isolation ensures the neural stream remains an unbiased, independent check, preventing memory-induced hallucination loops and error cascades across problems.

Without ground-truth labels or external supervision, ChemHarness substantially improves direct reasoning across all four open-source backbones evaluated. It outperforms ChemAgent by 23.3 and 22.9 points on Qwen2.5-7B and Qwen2.5-14B, and on Qwen2.5-32B and DeepSeek-V4.1-Flash it exceeds every published benchmark baseline, including ChemCrow (Bran et al., 2023) and ChemToolAgent (Yu et al., 2024). Our analysis demonstrates that cross-stream verification is markedly more precise than single-stream consistency, and that verifier isolation is essential to prevent error cascades into the evolving tool library.

Our contributions are the following.

- We identify chemical reasoning as a fundamental challenge for self-evolving harnesses due to the absence of ground-truth feedback, and we introduce dual-stream self-verification, which cross-verifies a neural stream against an executable symbolic stream at both outcome and step levels under a semantic judge to produce label-free supervision.

- We present ChemHarness, a closed-loop self-evolving harness coupling dual-stream self-verification with dual-scope memory evolution (a long-term procedural tool library admitted via self-constructed unit tests and a short-term episodic memory). ChemHarness enforces an asymmetric information flow that guarantees verifier isolation.

- We demonstrate that ChemHarness consistently boosts direct reasoning across four open-source backbones, outperforms representative chemical agents including ChemCrow and ChemAgent, and empirically verify that verifier isolation is indispensable for preventing error cascades during self-evolution.

2 Related Work

Chemical reasoning with language models. SciBench collects college-level problems in chemistry, physics, and mathematics and attributes the errors of language models to missing problem-solving skills (Wang et al., 2023b). StructChem organizes formula selection and step-by-step reasoning with confidence-guided review (Ouyang et al., 2023). ChemToolAgent studies tool use within a language agent for chemistry (Yu et al., 2024). ChemAgent maintains planning, execution, and knowledge memories built from a development set with solutions and reuses them for new problems (Tang et al., 2025). ChemCrow and Coscientist connect a language model to expert-written tools and to laboratory instruments (Bran et al., 2023; Boiko et al., 2023). ChemHarness uses no development data and evolves from the test problems alone.

Self-evolving agents. Voyager accumulates executable skills whose execution succeeds in its environment (Wang et al., 2023a). Agent Workflow Memory induces reusable workflows from past trajectories (Wang et al., 2024b), Buffer of Thoughts distills reusable thought templates (Yang et al., 2024), and ReasoningBank extracts strategies from successful and failed trajectories that the model itself judges (Ouyang et al., 2026). ACE evolves a context playbook through generation, reflection, and curation (Zhang et al., 2026). Tool-making methods enable a language model to write reusable functions. LATM has a strong model make tools that pass unit tests on labeled examples (Cai et al., 2024), CRAFT abstracts tools from the solutions of a training set and keeps those that solve their source problems (Yuan et al., 2024), and ReGAL refactors programs into shared functions that must reproduce the outputs of the original programs (Stengel-Eskin et al., 2024). TroVE induces a toolbox from a stream of tasks and selects among programs by the consistency of their executions (Wang et al., 2024a). On the chemistry problems of SciBench, tools created by such methods fall below direct programs unless a textbook guides their creation (Liu et al., 2026). ChemHarness admits a tool that the model abstracts only when the tool reproduces the values that dual-stream self-verification confirms, and its verifier remains independent of the memory that it builds.

Self-verification. Self-Consistency aggregates sampled reasoning paths of one stream (Wang et al., 2022). Mixture-of-thought cascades use the consistency between a chain-of-thought and a program-of-thought solution to decide when to defer to a stronger model (Yue et al., 2024). ReVeal trains a code agent with multi-turn reinforcement learning to verify its own programs with tool feedback (Jin et al., 2026). ChemHarness trains no parameters. It verifies at the level of intermediate steps as well as outcomes, and it turns the verification into the feedback that drives its self-evolution.

3 Methodology

Problem formulation. A large language model (LLM) $\mathcal{M}$ receives chemistry problems $q_1, q_2, \ldots, q_T$ one at a time in a fixed order. Each problem asks for one numerical value in a stated unit. No ground-truth answer, reward environment, reference solution, or development set is available at any time. A harness is the program surrounding a model that governs its inputs and the retention of its outputs (Lee et al., 2026). Because the parameters of $\mathcal{M}$ never change, any improvement over the sequence is attributable to the harness.

System overview. ChemHarness operates sequentially on an unlabelled problem stream $q_1, \ldots, q_T$ in a closed loop. For each problem q_t, the system first selects relevant procedural tools from the evolving library $\mathcal{T}$ and provides them exclusively to the symbolic stream. In each round $r \leq R$, the neural stream solves q_t via text derivation while the symbolic stream executes code. Once dual-stream verification succeeds, the cross-verified answer is returned, and the solution is abstracted into reusable tools validated via cross-verified unit tests. If all R rounds fail to reach consensus, an episodic memory aggregating verified lemmas and discrepancies conditions a final fallback solution. Figure 1 illustrates the closed loop from the input problem to the submitted answer. There are two tightly coupled components:

1. **Dual-stream self-verification and feedback extraction.** In every round of a problem, the model solves the problem once by a derivation in natural language and once by an executed program. Section 3.1 describes how cross-stream verification turns the two solutions into label-free feedback on final outcomes and on intermediate steps.

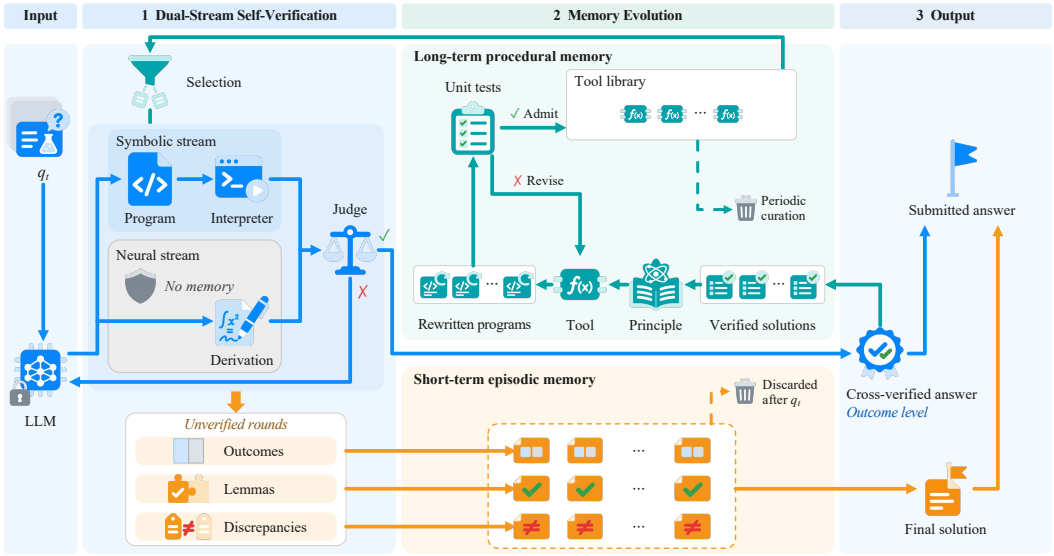

Figure 1: Overview of ChemHarness. Selected tools reach only the symbolic stream, and the short-term memory reaches only the final solution. The neural stream therefore receives no memory and remains an independent verifier.

2. **Memory evolution and asymmetric information flow.** Verified experience is consolidated into a procedural tool library across problems and an episodic short-term memory within a problem. Section 3.2 describes how the two memories return to the model under an asymmetric information flow that preserves the neural stream as an independent verifier.

3.1 Dual-Stream Self-Verification and Feedback Extraction

Errors in chemical reasoning arise from two sources, a conceptual oversight such as a missed implicit condition of an adiabatic process or a standard state, and a slip in the computation such as a unit conversion, an order of magnitude, or an arithmetic step. The two sources separate across modalities. A derivation in natural language computes in text and is prone to slips, whereas an executed program computes exactly but can encode a wrong formulation. Because the two solutions fail through different mechanisms, they rarely reach the same wrong value, whereas two samples of one stream tend to repeat the same mistake (Kim et al., 2025).

Dual-stream verification. In round $r \le R$ of problem q_t, the model $\mathcal{M}$ produces two solutions. In the neural stream $\mathcal{N}$, $\mathcal{M}$ writes a derivation d_r in natural language and carries out every computation in the text, which ends in the value $v_r^{\mathcal{N}}$. In the symbolic stream $\mathcal{S}$, $\mathcal{M}$ writes a Python program p_r, and an interpreter executes the program, which yields the value $v_r^{\mathcal{S}}$. The same model then acts as a judge that reads the problem and both solutions and returns $J_r \in \{0, 1\}$, whether the two solutions jointly establish the answer (Ouyang et al., 2026), and Appendix A lists its prompt. Round r is *verified*, $V_r = 1$, when the two values match within a relative tolerance and the judge confirms them,

$$V_r = \mathbb{I}\Big[\,|v_r^{\mathcal{N}} - v_r^{\mathcal{S}}| \le \epsilon \max\big(|v_r^{\mathcal{N}}|, |v_r^{\mathcal{S}}|\big)\Big] \cdot J_r = 1, \tag{1}$$

where $\epsilon = 0.05$ is the relative tolerance. Each round yields feedback at two levels, an answer at the outcome level and intermediate values at the step level.

Cross-verified answers. A cross-verified answer is the value that dual-stream self-verification establishes for a problem without any label. The rounds of problem q_t end at the first verified round, and the value of its neural stream becomes the cross-verified answer of q_t,

$$\hat{y}_t = v_{r^\star}^{\mathcal{N}}, \qquad r^\star = \min\{\, r \le R : V_r = 1 \,\}, \tag{2}$$

where $R = 8$ is the maximum number of rounds. A problem without a verified round is *unverified* and is answered by a final solution. Cross-verified answers also serve as the unit tests that admit tools into the tool library.

Lemmas. A lemma is an intermediate quantity that both streams of a round establish, and lemmas are kept even when the final values of the round differ. Let $\mathrm{Var}(p_r)$ be the set of variables of program p_r, u_x the value of variable x after execution, and $\mathrm{Num}(d_r)$ the set of numbers stated in derivation d_r. The lemmas of round r are

$$\mathcal{L}_r = \big\{\, (x, u_x) : x \in \mathrm{Var}(p_r),\ \exists\, y \in \mathrm{Num}(d_r),\ |u_x - y| \le \delta_{\text{tol}}\, |u_x| \,\big\}, \tag{3}$$

where δ_{tol} is a relative tolerance. Numbers given in the problem statement and integers up to ten are excluded, and Appendix B gives the matching rule.

Discrepancies. A discrepancy is a program variable that takes different values across the rounds of a problem. The discrepancies of problem q_t are

$$\mathcal{D}_t = \big\{\, (x, A_x) : |A_x| \ge 2 \,\big\}, \tag{4}$$

where A_x is the set of distinct values, up to the relative tolerance δ_{tol}, that variable x takes in the rounds of q_t. A discrepancy marks a step on which the rounds of a problem diverge. It records the competing values without deciding which of them is correct. Lemmas and discrepancies form the short-term memory of Section 3.2.

3.2 Memory Evolution and Asymmetric Information Flow

Long-term procedural memory. The tool library is the long-term procedural memory of ChemHarness and grows from verified problems. When a problem is verified, $\mathcal{M}$ reads the problem and the program of the verified round and names the principle that the solution applies, either a principle named on an earlier problem or a new one. Once a principle has at least $N_{\min}$ verified solutions, $\mathcal{M}$ writes a tool τ, whose code f_τ is a small set of documented Python functions. $\mathcal{M}$ also rewrites each source program i as a program p_i' that calls the tool, and the tool enters the library only when these rewritten programs reproduce the cross-verified answers v_i of their source problems,

$$\Big|\big\{\, i :\ \big|\mathrm{Exec}(f_\tau, p_i') - v_i\big| \le \delta_{\text{tol}}\, |v_i| \,\big\}\Big| \ \ge\ N_{\min}, \tag{5}$$

where $\mathrm{Exec}(f_\tau, p_i')$ is the value that the interpreter returns for p_i' with the tool defined. A tool that contains a number from the statement of a source problem is rejected, and after a failed check $\mathcal{M}$ reads the execution results and revises the tool.

Curation and selection. After every K_{cur} admitted tools, $\mathcal{M}$ reviews the library together with the usage record of each tool and removes the tools that another tool subsumes or that are too narrow to serve. The tool library after problem q_t is

$$\mathcal{T}_t = \{\tau_1, \ldots, \tau_{n_t}\}, \qquad \tau = (f_\tau,\ k_\tau,\ \mathcal{S}_\tau), \tag{6}$$

where n_t is the number of admitted tools, k_τ is the principle that tool τ computes, and $\mathcal{S}_\tau$ is the set of source problems whose cross-verified answers its rewritten programs reproduce. The library starts empty and persists across all later problems. For every new problem q_t, $\mathcal{M}$ reads the catalogue of the library and selects a subset $\mathcal{T}^\star$ of at most K_{ret} tools, which are defined in the interpreter and listed in the prompt of the symbolic stream.

Short-term episodic memory. Within a problem, the rounds that are not verified form the short-term memory

$$\mathcal{C}_t = (\mathcal{L}_t,\ \mathcal{D}_t,\ \mathcal{O}_t), \tag{7}$$

where $\mathcal{L}_t = \bigcup_r \mathcal{L}_r$ collects the lemmas of Equation (3), $\mathcal{D}_t$ holds the discrepancies of Equation (4) with their competing values, and $\mathcal{O}_t = \{(v_r^{\mathcal{N}}, v_r^{\mathcal{S}})\}_r$ contains the outcomes of both streams in these rounds. The short-term memory starts empty with each problem and is discarded when the problem ends. It is read only by the final solution, one additional derivation that rechecks the discrepancies and derives the answer again.

Asymmetric information flow. Self-evolution without labels depends on a verification signal that memory does not bias. If both streams read the same memory, they could copy the same wrong value from it and satisfy Equation (1) for that reason. ChemHarness therefore strictly partitions the information flow. The neural stream receives no memory, $d_r \sim \mathcal{M}(\cdot \mid q_t)$, and serves as an independent verifier. The symbolic stream is conditioned only on the selected tools, $p_r \sim \mathcal{M}(\cdot \mid q_t, \mathcal{T}^\star)$. A tool call is therefore verified only when the neural stream, which has no access to the tool, reaches the same value. The final solution is conditioned only on the short-term memory $\mathcal{C}_t$ and is sampled after all rounds have ended. Section 4.7 measures the effect of memory that reaches the neural stream.

4 Experiments

4.1 Setup

We evaluate on the chemistry problems of SciBench released with ChemAgent, 229 problems in four datasets, *chemmc*, *matter*, *atkins*, and *quan* (Wang et al., 2023b). We compare ChemHarness with direct reasoning, the few-shot baseline of SciBench (Wei et al., 2022), and with ChemAgent, which we run from its released code and configuration (Tang et al., 2025). Table 1 also lists the published results of StructChem, ChemCrow, ChemAgent, and ChemToolAgent (Ouyang et al., 2023; Bran et al., 2023; Yu et al., 2024). The backbones are Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct, and Qwen2.5-32B-Instruct (Qwen et al., 2025) and DeepSeek-V4.1-Flash. The tool library starts empty, and no ground-truth answer is used at any step. Answers are scored by the evaluation code of ChemAgent, and we report the unweighted average of the accuracy on the four datasets (Avg.), averaged over 3 runs. Appendix A lists the prompts, Appendix B.2 details decoding and scoring, and Appendix H the reproduction of ChemAgent. ChemAgent builds its library from a development set with solutions, and ChemHarness starts from an empty tool library without any solution.

Hyperparameter choices. All backbones and datasets share one configuration, and no hyperparameter is tuned for a particular model. A problem runs at most $R = 8$ rounds, a principle triggers the abstraction of a tool at $N_{\min} = 2$ verified solutions, lemmas, discrepancies, and unit tests use the relative tolerance $\delta_{\text{tol}} = 0.01$, the library is curated every $K_{\text{cur}} = 10$ admitted tools, and at most $K_{\text{ret}} = 4$ tools are selected for a problem. Appendix C.1 changes $N_{\min}$, δ_{tol}, K_{cur}, and K_{ret} in turn on Qwen2.5-7B, and the average accuracy varies by at most 2.7 points.

4.2 Main Results

Table 1 compares ChemHarness with StructChem, ChemCrow, ChemAgent, and ChemToolAgent, whose published results are obtained with GPT-4, GPT-4o, and Claude-3.5-Sonnet. On the open-weight Qwen2.5-14B, ChemHarness reaches an average accuracy of 63.3% and exceeds every published result except that of ChemToolAgent with Claude-3.5-Sonnet, as well as the 53.6% of ChemAgent with the five times larger Qwen2.5-72B. With Qwen2.5-32B, ChemHarness reaches 67.4% and exceeds all published results, and with DeepSeek-V4.1-Flash it achieves the highest accuracy on all four datasets, 80.1% on average.

These results require no curated resources. StructChem relies on manually designed structured instructions, ChemCrow and ChemToolAgent on 18 and 29 expert-designed tools, and ChemAgent on a memory library built from a development set with reference solutions, whereas ChemHarness starts from an empty tool library and uses no ground-truth answers. On identical backbones, ChemHarness outperforms the reproduced ChemAgent by 23.3 points with Qwen2.5-7B and 22.9 points with Qwen2.5-14B.

The advantage extends to individual datasets. With Qwen2.5-14B, ChemHarness surpasses every published system on *matter* and *atkins*, and with DeepSeek-V4.1-Flash it reaches 80.4% on *quan*, where no published system exceeds 61.6%. These datasets demand multistep numerical computation (Wang et al., 2023b), and ChemHarness checks each computed value against an independent derivation in the neural stream, whereas StructChem revises its solution within a single stream and tool-augmented agents fall below their base models on such questions (Yu et al., 2024).

Table 1: Accuracy (%) of ChemHarness and published agents and harnesses on the four SciBench chemistry datasets. Rows marked † report published results, with Avg. recomputed from their per-dataset results, and rows marked ‡ run the released code and configuration of ChemAgent. Table 2 reports the standard deviations of ChemHarness. The best entry of each column is in bold.

Method	*chemmc*	*matter*	*atkins*	*quan*	Avg.
StructChem† (GPT-4)	59.0	30.6	59.8	41.2	47.6
ChemCrow† (GPT-4o)	60.5	34.0	58.4	50.5	50.9
ChemCrow† (Claude-3.5-Sonnet)	65.8	41.8	59.4	54.5	55.4
ChemAgent† (GPT-4)	74.4	49.0	61.2	44.1	57.2
ChemAgent† (Qwen2.5-72B)	69.2	44.9	56.1	44.1	53.6
ChemAgent‡ (Qwen2.5-7B)	31.6	26.5	27.1	16.7	25.5
ChemAgent‡ (Qwen2.5-14B)	48.7	41.5	42.1	29.4	40.4
ChemToolAgent† (GPT-4)	70.1	54.4	65.1	45.1	58.7
ChemToolAgent† (GPT-4o)	64.9	56.0	63.8	48.5	58.3
ChemToolAgent† (Claude-3.5-Sonnet)	79.8	56.7	66.3	61.6	66.1
ChemHarness (Qwen2.5-7B)	72.6	36.7	50.5	35.3	48.8
ChemHarness (Qwen2.5-14B)	70.9	60.5	68.8	52.9	63.3
ChemHarness (Qwen2.5-32B)	74.4	63.9	71.7	59.8	67.4
ChemHarness (DeepSeek-V4.1-Flash)	**83.8**	**74.8**	**81.3**	**80.4**	**80.1**

4.3 ChemHarness Improves Every Backbone

Table 2: Average accuracy (%) of direct reasoning, the first-round neural stream of ChemHarness with three demonstrations, and of ChemHarness, with the standard deviation (SD), the gain of ChemHarness, and the p-value of a paired sign test over problems.

	Direct reasoning		ChemHarness			
Backbone	Avg.	SD	Avg.	SD	Gain	p
Qwen2.5-7B	36.1	4.0	48.8	1.3	+12.7	<0.001
Qwen2.5-14B	52.3	3.0	63.3	0.4	+11.0	<0.001
Qwen2.5-32B	55.8	1.1	67.4	0.4	+11.6	<0.001
DeepSeek-V4.1-Flash	73.7	0.4	80.1	1.6	+6.4	<0.001

Table 2 compares ChemHarness with direct reasoning on the same backbones. ChemHarness improves the average accuracy of every backbone, from 6.4 points with DeepSeek-V4.1-Flash to 12.7 points with Qwen2.5-7B. A paired sign test over problems yields p <0.001 on every backbone.

The improvement arises on the problems that remain unverified after the first round. A problem verified in the first round receives the first-round derivation of the neural stream as its cross-verified answer, which coincides with the answer of direct reasoning. On the remaining problems, direct reasoning is correct on 25.4% with Qwen2.5-7B and 47.2% with DeepSeek-V4.1-Flash, whereas ChemHarness attains 41.6% and 67.1%. A stronger backbone verifies more problems in the first round, 23.0% with Qwen2.5-7B against 69.3% with DeepSeek-V4.1-Flash, and leaves fewer problems for further rounds to improve. The smaller gain of DeepSeek-V4.1-Flash follows from the larger first-round share. Appendix D analyzes the improvement by round.

4.4 How ChemHarness Evolves across Problems

ChemHarness starts with an empty tool library and receives the problems one at a time. Figure 2 traces the library and the accuracy along the problem sequence. The library grows on every backbone throughout the sequence and holds 31 tools with Qwen2.5-7B and 81 tools with DeepSeek-V4.1-Flash after the last problem. The net gain over direct reasoning rises at a steady rate on every backbone. The improvement of Table 2 is therefore spread over the whole sequence instead of a few problems. On every backbone, ChemHarness corrects more answers of direct reasoning than it turns incorrect, and the ratio is highest with DeepSeek-V4.1-Flash, at 47 against 5. In two random orders of the

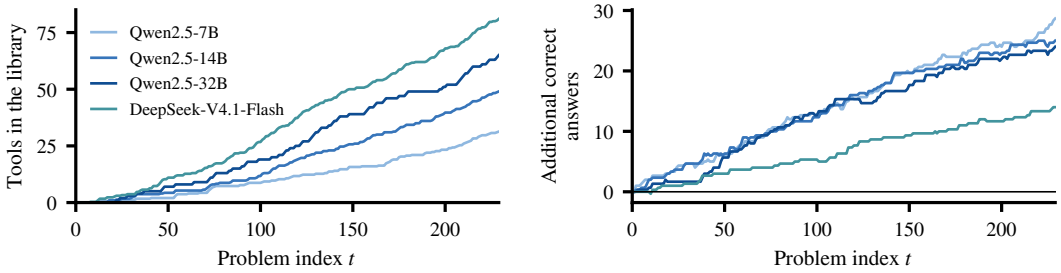

Figure 2: Tools in the library (left) and additional correct answers of ChemHarness over direct reasoning (right) along the fixed order of the 229 test problems, which interleaves the four datasets.

Table 3: Avg. (%) of ChemHarness and of variants that each remove one component. Without cross-verified answers, every problem is answered by its final solution, and without the short-term memory, the final solution of an unverified problem is conditioned on no memory. Appendix C reports the accuracy on each dataset.

Variant	Qwen2.5-7B	Qwen2.5-14B	Qwen2.5-32B	DeepSeek-V4.1-Flash
ChemHarness	49.9	63.3	67.1	81.4
without cross-verified answers	38.3	57.0	64.1	78.3
without the short-term memory	44.4	63.4	66.6	79.1

problems, ChemHarness exceeds direct reasoning by at least 5.2 points on every backbone evaluated, and Appendix C.2 reports the accuracy in each order.

The unit tests admit 38% of the abstractions of Qwen2.5-7B, 47% of Qwen2.5-14B, 55% of Qwen2.5-32B, and 71% of DeepSeek-V4.1-Flash. An admitted tool computes a principle from its inputs to its result, such as the series-limit wavelength of a hydrogen spectral series. A program calls a selected tool on 33% of the problems given tools with DeepSeek-V4.1-Flash and on 18% with Qwen2.5-7B. Appendix E analyzes the abstraction, the curation, and the use of the tools on every backbone.

4.5 What Each Component Contributes

Table 3 removes one component of ChemHarness at a time. Removing cross-verified answers lowers the average accuracy on every backbone, by up to 11.6 points with Qwen2.5-7B. Removing the short-term memory lowers the accuracy by 5.5 points with Qwen2.5-7B and 2.3 points with DeepSeek-V4.1-Flash and changes it by at most 0.5 points with Qwen2.5-14B and Qwen2.5-32B. Section 4.7 isolates the effect of the short-term memory on the problems that it conditions.

The two components act on different problems. On verified problems, the first-round cross-verified answer is correct in 74% to 86% of the cases across backbones, whereas on unverified problems the final solution is correct in only 24% to 34%. Removing cross-verified answers therefore replaces the most accurate answers with final solutions. The short-term memory conditions only the final solutions of unverified problems, which constitute 44% of the problems of Qwen2.5-7B but only 6% of those of DeepSeek-V4.1-Flash, and its removal accordingly has the largest effect on Qwen2.5-7B.

4.6 Cross-Stream Verification Outperforms Single-Stream Consistency

Table 4 reports the precision of matching outcomes on problems with at least two rounds. Dual-stream verification reaches a precision between 63.0% and 75.4% across backbones, against 46.2% to 55.7% for two rounds of the same stream. The gap is largest with Qwen2.5-14B, at 66.7% against 46.2%. On each of the four datasets and with every backbone, verified problems are also more accurate than unverified ones, as Appendix D.3 shows. Cross-stream verification also provides step-level feedback when the outcomes of a round differ. Across backbones, 11% to 27% of such rounds contain a new lemma, and Appendix D.5 reports the results for each backbone.

Table 4: Precision (%) of matching outcomes on problems with at least two rounds, for the two streams of one round and for two rounds of one stream.

Backbone	Dual-stream	Within one stream
Qwen2.5-7B	63.0	55.7
Qwen2.5-14B	66.7	46.2
Qwen2.5-32B	68.3	50.8
DeepSeek-V4.1-Flash	75.4	54.2

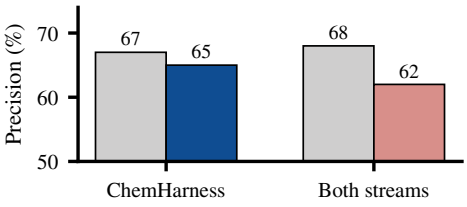

Figure 3: Precision of cross-verified answers without (grey) and with (colored) lemmas.

4.7 Why the Verifier Receives No Memory

Memory that reaches the neural stream lowers the precision of cross-verified answers. Figure 3 compares two flows of the short-term memory on Qwen2.5-7B and Qwen2.5-14B, each against the same problems solved without memory. When the lemmas condition both streams, the precision of cross-verified answers decreases from 68% to 62%. When they condition only the final solution, as in ChemHarness, the precision changes from 67% to 65%, within sampling variation. A wrong value that both streams copy from memory becomes a cross-verified answer and later a unit test for new tools.

Within the final solution, the short-term memory corrects unverified problems. Unverified problems account for 58% of the wrong answers of ChemHarness without the short-term memory. A counterfactual replay, detailed in Appendix C.4, reuses every recorded round and judgment and samples only the final solution again with the short-term memory. On 458 unverified problems of Qwen2.5-7B, Qwen2.5-14B, and Qwen2.5-32B, the final solution is correct on 18.6% without the short-term memory and on 29.3% with it, a net gain of 49 problems.

5 Conclusion

We have presented ChemHarness, a closed-loop self-evolving harness for chemical reasoning without external ground truth, reference solutions, or parameter updates. By cross-verifying a neural stream against a symbolic stream, ChemHarness turns matching final answers and intermediate values into rich, label-free feedback. Furthermore, strictly isolating the neural stream as an unconditioned verifier prevents hallucinated consensus and error cascades across procedural tool libraries and episodic memories. Across multiple open-source backbones, ChemHarness consistently improves direct reasoning and outperforms existing chemical agent baselines.

6 Limitations

ChemHarness requires problems whose answers can be computed by a short program, which holds for the quantitative chemistry problems of SciBench. The judge and both streams use the same backbone, and a verified outcome can still be wrong when both streams make the same error. The short-term memory acts only on unverified problems. The tool library requires a backbone that applies the abstracted tools reliably.

Reproducibility Statement

Appendix A lists the prompts, Appendix B specifies the harness, models, decoding, runs, and scoring, and Appendix H describes the ChemAgent reproduction. Code, problem-level outputs, and tool libraries will be released.

AI Assistance Disclosure

In this work, we used generative AI tools to edit the paper for readability. We have reviewed all AI-assisted text, and we take responsibility for the final content of this work, including text produced with the aid of generative AI.

References

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, volume 2024, pp. 54067–54089, 2024.

Gabriel Grand, Lio Wong, Maddy Bowers, Theo X Olausson, Muxin Liu, Joshua B Tenenbaum, and Jacob Andreas. Lilo: Learning interpretable libraries by compressing and documenting code. In *International Conference on Learning Representations*, volume 2024, pp. 30399–30446, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

Yiyang Jin, Kunzhao Xu, Hang Li, Xueting Han, Yanmin Zhou, Cheng Li, and Jing Bai. Reveal: Self-evolving code agents via reliable self-verification. In *International Conference on Learning Representations*, volume 2026, pp. 149800–149824, 2026.

Elliot Kim, Avi Garg, Kenny Peng, and Nikhil Garg. Correlated errors in large language models. *arXiv preprint arXiv:2506.07962*, 2025.

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

Xiao Liu, Da Yin, Zirui Wu, and Yansong Feng. Reftool: Reference-guided tool creation for knowledge-intensive reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 130663–130686, 2026.

Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, et al. A framework for evaluating the chemical knowledge and reasoning abilities of large language models against the expertise of chemists. *Nature Chemistry*, 17(7):1027–1034, 2025.

Siru Ouyang, Zhuosheng Zhang, Bing Yan, Xuan Liu, Yejin Choi, Jiawei Han, and Lianhui Qin. Structured chemistry reasoning with large language models. *arXiv preprint arXiv:2311.09656*, 2023.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, and Udaya Ghai. Mind the gap: Examining the self-improvement capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 39894–39931, 2025.

Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. Regal: Refactoring programs to discover generalizable abstractions. *arXiv preprint arXiv:2401.16467*, 2024.

Xiangru Tang, Tianyu Hu, Muyang Ye, Daniel Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, et al. Chemagent: Self-updating memories in large language models improves chemical reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 100922–100958, 2025.

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

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*, 2023b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024a.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024.

Botao Yu, Frazier N Baker, Ziru Chen, Garrett Herb, Boyu Gou, Daniel Adu-Ampratwum, Xia Ning, and Huan Sun. Chemtoolagent: The impact of tools on language agents for chemistry problem solving. *arXiv preprint arXiv:2411.07228*, 2024.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, volume 2024, pp. 40097–40125, 2024.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thought representations for cost-efficient reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 21691–21728, 2024.

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

A PROMPTS

Every prompt below is quoted verbatim, with placeholders in brackets, in the order in which a problem uses it.

The neural stream is instantiated with the few-shot chain-of-thought format of SciBench, and the symbolic stream with its few-shot Python format. Each conversation contains a system message, three demonstrations from the dataset of the problem, and the target problem. The prompt of the symbolic stream receives the same demonstrations with their Python solutions. The system messages are as follows.

Neural stream, system message

Please provide a clear and step-by-step solution for a scientific problem in the categories of Chemistry, Physics, or Mathematics. The problem will specify the unit of measurement, which should not be included in the answer. Express the final answer as a decimal number with three digits after the decimal point. Conclude the answer by stating "The answer is therefore `\boxed{[ANSWER]}`."

Symbolic stream, system message

Please provide a clear and step-by-step solution for a scientific problem in the categories of Chemistry, Physics, or Mathematics. The problem will specify the unit of measurement. Please translate the solution steps into Python code and encase the Python code within triple backticks for clarity.

The judge receives the problem and the latest derivation and program, truncated to 3000 characters for the derivation, 2500 characters for the program, and 300 characters for the program output.

Judge

System. You review draft solutions to a chemistry problem. Reply in English, with JSON only.

User. Problem: `[problem]`
— Round `[r]` —
[Written derivation] `[derivation]`
[Derivation's answer] `[value]`
[Program] `[program]`
[Program output] `[output]`
[Program's answer] `[value]`
[The two answers agree within 5%: `[yes|no]`]
Question for you: are these drafts enough to settle the problem (the reasoning is sound and the two forms support each other), or is another independent round needed? Answer with JSON: `{"enough": true or false, "reason": "<one sentence>"}`

A problem that remains unverified after 8 rounds is answered by the final solution, which uses the few-shot conversation of the neural stream extended by one instruction and by the short-term memory of the problem when it is not empty. When the answer contains a Python block, the block is executed and its output is returned in a second turn.

Final solution

Appended to the target problem. Give the final answer in the unit stated in the problem, as in the examples, and conclude with "The answer is therefore `\boxed{[ANSWER]}`."

Appended after the instruction when the short-term memory is not empty. Intermediate results for this problem that were obtained independently by both a written derivation and a program in earlier attempts. You may build on them if they fit your solution.
– `[name] = [value]` (derivation: "`[derivation line]`"), one line per verified step, at most eight

Appended after that when the rounds disagree on some quantity. Quantities on which the earlier attempts for this problem disagree. The attempts did not settle them, so check these steps carefully instead of

Final solution (continued)

copying any of the values.
- `[name]: [value] ([k] attempts[, also stated by a written derivation]); ...`, one line per disputed quantity, at most six
- `final answer: written derivations gave [value] ([k] attempts); ...; programs gave [value] ([k] attempts); ...`

Second turn, when the answer contains code. Your Python code was executed. Output: `[output]` Using this output, state the final answer in the unit stated in the problem and conclude with "The answer is therefore `\boxed{[ANSWER]}`."

The tool library adds four prompts. Every problem starts with the selection of tools. The prompt lists the catalogue of the tool library, one line per function with its signature and the first line of its docstring, grouped by principle. The selected tools are defined in the Python interpreter before every program of the problem runs, and the prompt of the symbolic stream lists them after the target problem.

Selection of tools

Problem: [problem]
A library of Python tools for chemistry calculations, grouped by category: [catalogue]
Which tools would help a program solve this problem? First decide which categories are relevant, then choose at most four tools from them. Choose none if no tool applies. Reply with JSON only: {"categories": ["<category>", ...], "tools": ["<tool name>", ...]}

Selected tools in the symbolic stream

Tools already defined for you. They were written from earlier problems whose results were verified independently. Call a tool when it computes what this problem needs, with this problem's values in the units its docstring states; do not redefine it.
`[source code of the selected tools]`

After a problem is verified, the model names its principle and, when the principle has enough verified solutions, abstracts them into a tool. After every 10 admitted tools, the model curates the tool library.

Principle of a verified solution

Below is a chemistry problem and a program whose result an independent written derivation confirmed.
[problem] Verified program: [program] Verified result: [value]
Name the chemical principle or method that this solution applies, as a short category name that also fits other problems of the same kind. Existing categories: [one line per principle with its description]
If one of them fits, reuse its exact name; otherwise create a new one. Reply with JSON only: {"category": "<name>", "description": "<one sentence on what problems of this category compute>"}

Abstraction of verified solutions into a tool

You maintain a library of reusable Python tools for chemistry calculations. The problems below belong to the category '[principle]'. For each problem, a program computed a result that an independent written derivation confirmed.
Problem [k] [problem] Verified program: [program] Verified result: [value] (one block per solution)
Existing tools in this category. You may improve them, generalize them, or add new ones: [source code] (when the principle has tools)
Write one to three general tools that capture the principle these problems share, so that a future problem of this category can be solved by calling them. Requirements:
- Each tool is a Python function with a descriptive snake_case name and a docstring: one line on what it computes and when it applies, then one line per parameter and one for the return value, each with its physical meaning and unit.
- Every quantity that differs between problems is a parameter. Physical constants may appear as named constants inside the function.
- A tool carries out the whole computation from the principle to the result, not a single arithmetic step.
- Do not use any number that appears only in one of the problem statements.
Then, for each problem, write a short program that solves it by calling your tools with that problem's

Abstraction of verified solutions into a tool (continued)

values and prints only the final result.
Format: first one python block with all tools (and their imports), then for each problem a line "### Program k" followed by one python block.

When the unit tests of a tool fail, the conversation continues with the message *Running your programs with your tools failed the check:* followed by one line per failed program, and *Each program must call the tools and print the verified result. Revise the tools and the programs, in the same format.*

Maintenance of the tool library

You maintain a library of Python tools for chemistry calculations. Each line lists a tool, its category, its description, and how it has served: how often it was offered for a problem, how often a program called it, and how often a call ended in a verified solution. [one line per tool]
Decide which tools to remove. Remove a tool when another tool computes the same thing more generally, or when it is too narrow or unhelpful to keep. Keep every tool that serves a distinct purpose. Reply with JSON only: {"remove": [{"name": "<tool>", "reason": "<short reason>"}]} (an empty list if nothing should be removed).

B Implementation Details

B.1 Harness

A problem runs at most $R = 8$ rounds. The two outcomes of a round match within the relative tolerance $\epsilon = 0.05$. Lemmas, discrepancies, and the unit tests of tools use the relative tolerance $\delta_{\text{tol}} = 0.01$.

The Python interpreter prints every module-level numerical variable of a program after it runs. The numbers written in the derivation of the neural stream of the same round are parsed, including scientific notation. Zero, integers up to ten, and numbers that appear in the problem statement are excluded on both sides. A program variable is a lemma when some remaining number in the derivation is within one percent of its value. The line of the derivation whose number is closest to the value is stored with the step.

A verified problem contributes its program and the value of that program in the verified round. The model names a principle for every verified problem, and a principle triggers an abstraction once it accumulates $N_{\min} = 2$ verified solutions, and a tool is admitted when at least $N_{\min}$ rewritten programs reproduce their cross-verified answers. After a failed check, the model revises the tool for at most 2 further attempts, and an admitted tool that redefines a function of an earlier tool replaces the earlier tool. The abstraction reads at most 4 solutions of the principle, the newest first. The rewritten programs run in the same interpreter as the symbolic stream, and a rewritten program passes when its printed result differs from the verified value by at most a relative δ_{tol} and it calls one of the new functions. A tool is rejected when it contains a number within 0.2% of a number in the statement of a source problem, other than zero and integers up to ten. At most $K_{\text{ret}} = 4$ tools are selected for a problem, the library is curated after every $K_{\text{cur}} = 10$ admitted tools, and the catalogue lists at most 150 functions, those whose calls most often ended in a verified round first.

The lemmas of every round that is not verified are added to the short-term memory with the variable name, its value, and the line of the derivation whose number is closest to the value. A value that is already in the memory is not added again. The final solution receives the eight most recent entries. A program variable is a discrepancy when two of its values over the rounds differ by more than one percent. Values given in the problem and small integers are excluded, as for lemmas. For each discrepancy, the final solution receives up to three distinct values, ordered by the number of rounds that reached them, for at most six quantities. The final answers of the two streams are grouped by value in the same way.

B.2 Experimental Protocol

The test set released with ChemAgent contains 229 problems in four datasets, *chemmc* (39), *matter* (49), *atkins* (107), and *quan* (34). Each problem asks for one numerical value in a stated unit. Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct, and Qwen2.5-32B-Instruct are served with vLLM in bfloat16. Every backbone samples with its default decoding parameters, and the derivation and the program each generate at most 4096 tokens. For Qwen2.5 the defaults are temperature 0.7, top-p 0.8, top-k 20, and repetition penalty 1.05, which are passed explicitly. The judge and the final solution use the same model with at most 400 and 3000 generated tokens. DeepSeek-V4.1-Flash runs with the same prompts and limits through the chat interface of its provider.

ChemHarness and direct reasoning run three times with different sampling seeds on Qwen2.5-7B, Qwen2.5-14B, Qwen2.5-32B, and DeepSeek-V4.1-Flash. Ablations, information flows, transfers, hyperparameter settings, and random problem orders run once, with the first seed. The sampling seed of every call is derived from the seed of the run, the problem, the stream, and the round. Problems are processed in a fixed order with four problems processed concurrently, and a problem selects from the tools admitted before its selection. Table 2 reports, for every backbone, the average accuracy of direct reasoning and ChemHarness with its standard deviation over the 3 runs, the gain of ChemHarness, and the p-value of a paired sign test over problems. The reproduction of ChemAgent varies by 3.1 and 0.9 points over its runs with Qwen2.5-7B and Qwen2.5-14B. All other results in the appendix are means or totals over the three runs.

Answers are scored by the `evaluate_no_llm` function released with ChemAgent. It accepts a response when any number in the response is within a relative tolerance of 5% of the ground-truth answer (10% for values below one), and it also compares the leading digits of the number in scientific notation. Every method is scored on the single number of its final answer, extracted by one rule. The answer is the content of the last `\boxed{}` in the response, or the text after the last occurrence of "answer is", or the response itself when it is a single short value such as a program output. Its first number, with an explicit factor 10^b applied and a fraction `\frac{a}{b}` evaluated, is passed to the function. A response without such a number is counted as wrong. The same rule scores ChemHarness, its ablations, direct reasoning, and the ChemAgent reproduction.

C Robustness of Components and Memory Flows

Table 5: Accuracy (%) of ChemHarness and its ablations on each dataset, as in Table 3.

Backbone	Variant	*chemmc*	*matter*	*atkins*	*quan*	Avg.
Qwen2.5-7B	ChemHarness	74.4	36.7	53.3	35.3	49.9
	without cross-verified answers	59.0	30.6	40.2	23.5	38.3
	without the short-term memory	69.2	32.7	52.3	23.5	44.4
Qwen2.5-14B	ChemHarness	71.8	59.2	69.2	52.9	63.3
	without cross-verified answers	69.2	53.1	64.5	41.2	57.0
	without the short-term memory	69.2	63.3	71.0	50.0	63.4
Qwen2.5-32B	ChemHarness	74.4	63.3	72.0	58.8	67.1
	without cross-verified answers	74.4	57.1	69.2	55.9	64.1
	without the short-term memory	79.5	59.2	74.8	52.9	66.6
DeepSeek-V4.1-Flash	ChemHarness	84.6	73.5	82.2	85.3	81.4
	without cross-verified answers	82.1	73.5	81.3	76.5	78.3
	without the short-term memory	82.1	73.5	81.3	79.4	79.1

Table 5 reports the ablations of Table 3 on each dataset, which traces the effect of each component to the problems it changes. Without cross-verified answers, the accuracy drops on every dataset of Qwen2.5-7B and Qwen2.5-14B. The short-term memory of Qwen2.5-7B carries its largest effect on quan, followed by chemmc.

Table 6: Hyperparameter sensitivity of ChemHarness on Qwen2.5-7B, accuracy (%) on each dataset. Each row changes one hyperparameter of the default configuration ($N_{\min} = 2$, $\delta_{\text{tol}} = 0.01$, $K_{\text{cur}} = 10$, $K_{\text{ret}} = 4$). Δ is the change of the average accuracy from the default.

Setting	*chemmc*	*matter*	*atkins*	*quan*	Avg.	Δ
Default	74.4	36.7	53.3	35.3	49.9	–
$N_{\min} = 3$	71.8	46.9	53.3	26.5	49.6	-0.3
$\delta_{\text{tol}} = 0.02$	74.4	36.7	54.2	23.5	47.2	-2.7
$K_{\text{cur}} = 20$	74.4	38.8	56.1	35.3	51.1	$+1.2$
$K_{\text{ret}} = 8$	74.4	36.7	55.1	29.4	48.9	-1.0

C.1 Hyperparameter Sensitivity

Table 6 changes one hyperparameter of the default configuration at a time on Qwen2.5-7B. The average accuracy stays within one standard deviation across runs (1.3 points) when $N_{\min}$ rises to 3, K_{cur} to 20, or K_{ret} to 8. Loosening δ_{tol} to 0.02 allows values that differ by up to 2% to count as the same lemma and pass the unit tests of a tool, and it lowers the average accuracy by 2.7 points, all of it on *quan*. The results therefore do not rest on a fine choice of these values, and the tolerance of verification is the one that matters.

C.2 Order of the Problem Sequence

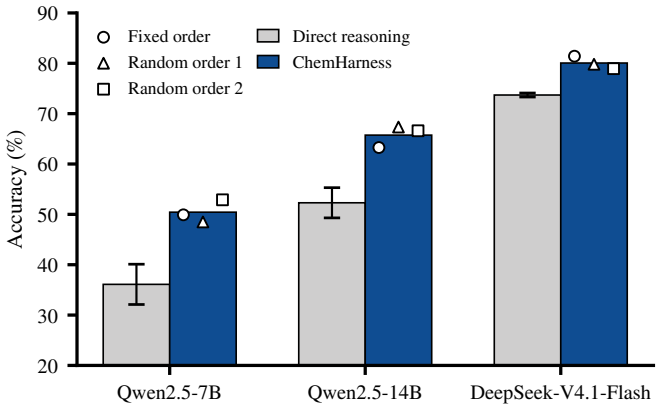

Figure 4: Accuracy of direct reasoning and of ChemHarness in the fixed order of the test problems and in two random orders. Each bar of ChemHarness is the mean of the three orders, and the markers give the accuracy in each order.

ChemHarness processes the test problems sequentially, and the tool library available to a problem is determined by the problems that precede it. We therefore evaluate ChemHarness on Qwen2.5-7B, Qwen2.5-14B, and DeepSeek-V4.1-Flash in two random orders of the test problems as well as in the fixed order of the main results. Figure 4 reports the accuracy in the three orders together with the accuracy of direct reasoning. The improvement over direct reasoning does not depend on the order. In every order, ChemHarness exceeds direct reasoning by at least 12.4 points with Qwen2.5-7B, 11.0 points with Qwen2.5-14B, and 5.2 points with DeepSeek-V4.1-Flash. The order changes the accuracy by at most 4.4 points, less than half of the improvement on each backbone.

C.3 Information Flows of Memory

Figure 5 compares four information flows of memory. Each flow changes which part of the harness one memory conditions and keeps every other part fixed. In the flow of ChemHarness for tools, the selected tools are defined in the Python interpreter and listed in the prompt of the symbolic stream. When tools reach both streams, the symbolic stream receives the tools in the same way, and the prompt of the neural stream lists the same tools with the instruction to apply their computations by

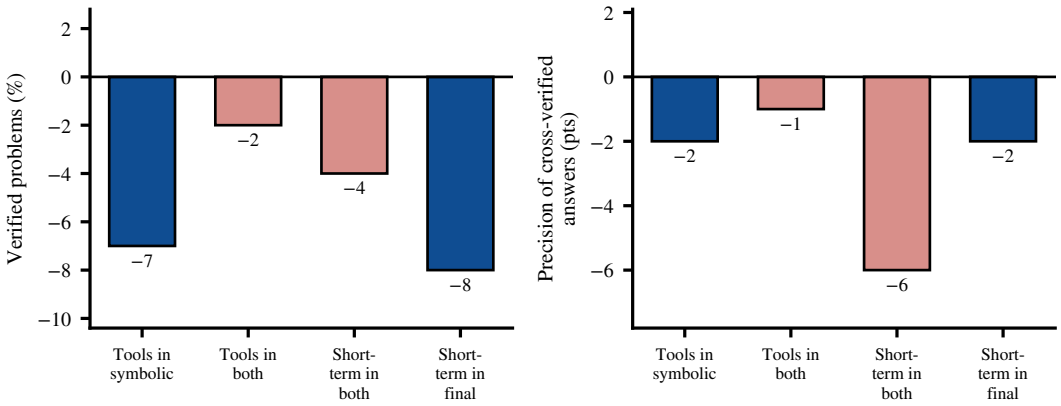

Figure 5: Change in the number of verified problems and in the precision of their cross-verified answers for four information flows of memory, relative to runs without memory on the same problems. Qwen2.5-7B and Qwen2.5-14B. The two flows of ChemHarness are in blue.

hand. When the short-term memory reaches both streams, the prompts of both streams list the lemmas of earlier rounds of the problem from the second round on, whereas in the flow of ChemHarness the short-term memory conditions only the final solution. Flows of tools are measured on the problems for which tools are selected. Flows of the short-term memory are measured on the problems that are not verified in the first round. Every flow is compared with the matching runs without memory, and for each problem the comparison uses the mean over those runs. The flows pool 2, 2, 2, and 2 runs of Qwen2.5-7B and Qwen2.5-14B, which cover 269, 262, 293, and 290 problem instances.

C.4 Counterfactual Replay of the Short-Term Memory

Two runs with the same sampling configuration sample different rounds, because batched inference does not reproduce a sample exactly. The comparison of two independent runs therefore mixes the effect of a component with the variation between runs. The short-term memory conditions only the final solution, which admits a counterfactual replay. We load the recorded requests and responses of a run without the short-term memory, replay every round and every judgment from the record, and intervene only on the final solution of each unverified problem, which is sampled again conditioned on the short-term memory. Problems with an empty short-term memory keep their recorded final solution. A problem is left out when a recorded request cannot be reproduced exactly, which happens when a program prints a different value on re-execution. The 7 counterfactual replays cover three runs each of Qwen2.5-7B and Qwen2.5-14B, which run without any memory, and one run of Qwen2.5-32B, which retains its tool library but omits the short-term memory. Over these replays, the short-term memory turns 80 wrong answers into correct ones and 31 correct answers into wrong ones.

D Verification

D.1 Rounds of Verification and the Source of the Gain

Figure 6 shows when problems are verified. The share verified in the first round grows with the backbone, from 23% with Qwen2.5-7B to 69% with DeepSeek-V4.1-Flash, and the share of unverified problems falls from 44% to 6%. Cross-verified answers of the first round are the most accurate, between 74% and 86%.

The gain of ChemHarness arises on the problems that remain unverified after the first round. A problem verified in the first round receives the first-round derivation of the neural stream as its cross-verified answer, which is also the answer of direct reasoning. Figure 7 separates the test problems at this point. On problems verified in the first round, ChemHarness and direct reasoning have the same accuracy. On the remaining problems, direct reasoning is correct on 25.4% with Qwen2.5-7B and 47.2% with DeepSeek-V4.1-Flash, and ChemHarness raises this to 41.6% and 67.1%. The first round

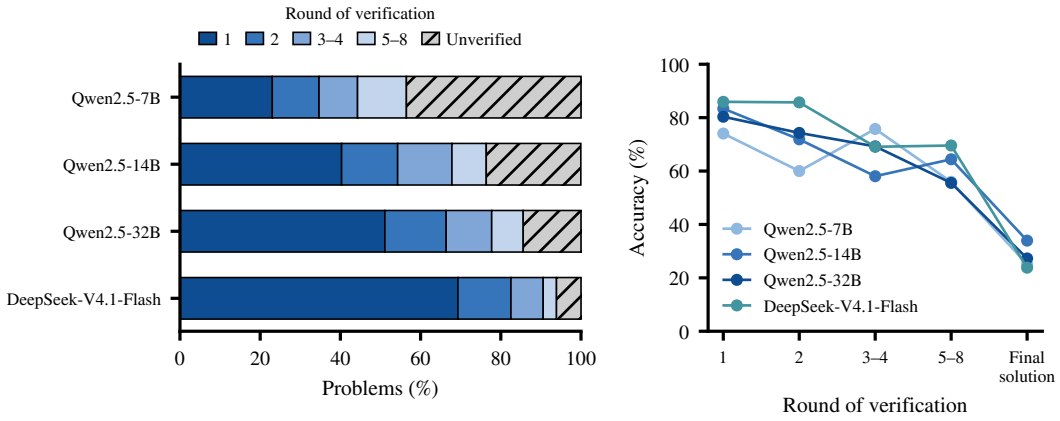

Figure 6: Left: share of the test problems verified in each round, and share of unverified problems. Right: accuracy of the cross-verified answers of each round, and of the final solution of unverified problems. Rounds with fewer than five verified problems are omitted.

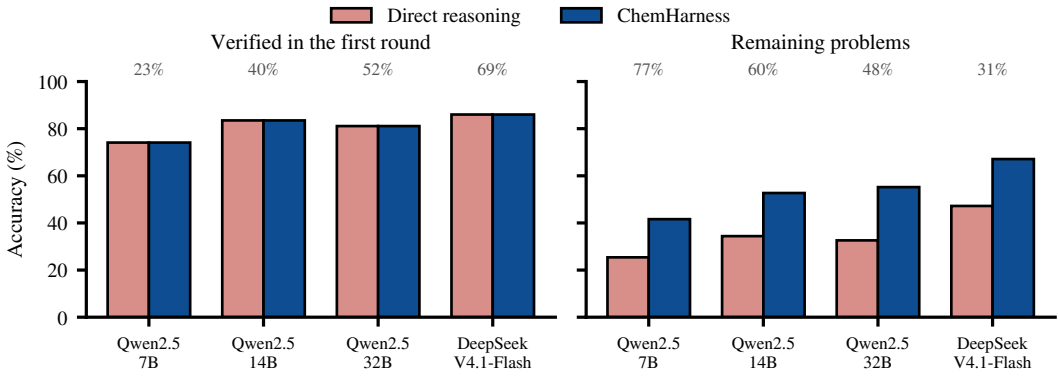

Figure 7: Accuracy of direct reasoning and ChemHarness on the problems verified in the first round (left) and on the remaining problems (right). The share of problems in each part is printed above each backbone.

therefore separates problems that a backbone already solves from problems on which it needs further rounds.

The size of the gain depends on how many problems remain after the first round and how hard they are. DeepSeek-V4.1-Flash verifies 69.3% of the problems in the first round, against 23.0% for Qwen2.5-7B and 40.3% for Qwen2.5-14B. A stronger backbone therefore leaves fewer problems for further rounds, and those problems are harder, since direct reasoning is correct on only 47.2% of the remaining problems of DeepSeek-V4.1-Flash. ChemHarness raises this share to 67.1%.

D.2 Fixed and Broken Problems

Figure 8 counts the answers of direct reasoning that ChemHarness corrects and turns incorrect, and Section 4.4 traces the cumulative difference between ChemHarness and direct reasoning along the problem sequence. On every backbone, ChemHarness corrects more answers of direct reasoning than it turns incorrect, and the net gain grows steadily along the sequence. The gain of Table 2 therefore does not stem from a few problems or from one part of the sequence.

D.3 Verification on Each Dataset

Figure 9 separates verified problems from the rest on each dataset. On every dataset and with every backbone, verified problems are more accurate than unverified ones. Dual-stream self-verification

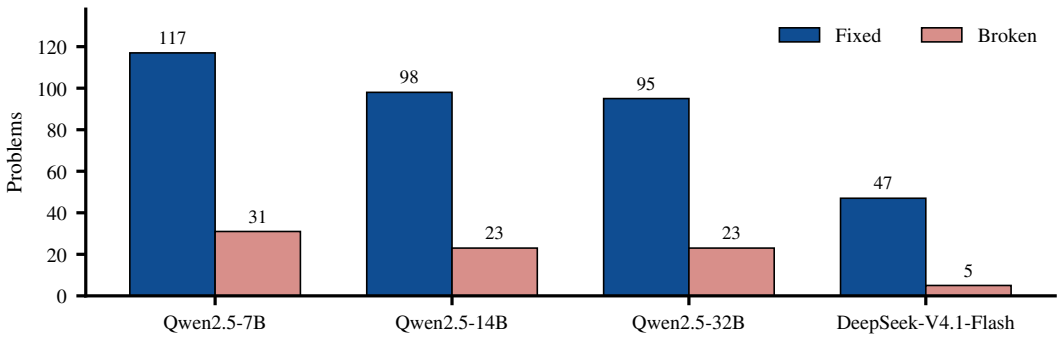

Figure 8: Problems that ChemHarness answers correctly and direct reasoning answers wrongly (fixed), and the reverse (broken).

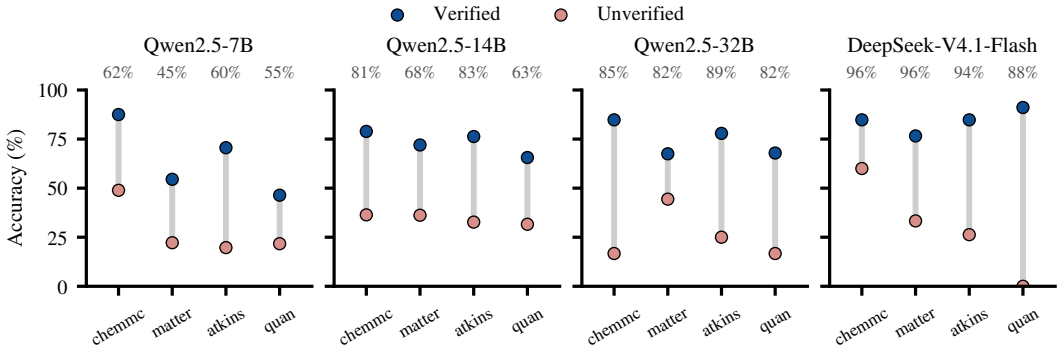

Figure 9: Accuracy of verified and unverified problems on each dataset. The number above each dataset is the share of its problems that are verified.

thus separates the two groups without any label on all four datasets. The share of verified problems is lowest where the backbone is weakest, on *matter* and *quan* with Qwen2.5-7B.

D.4 Discrepancies of Unverified Problems

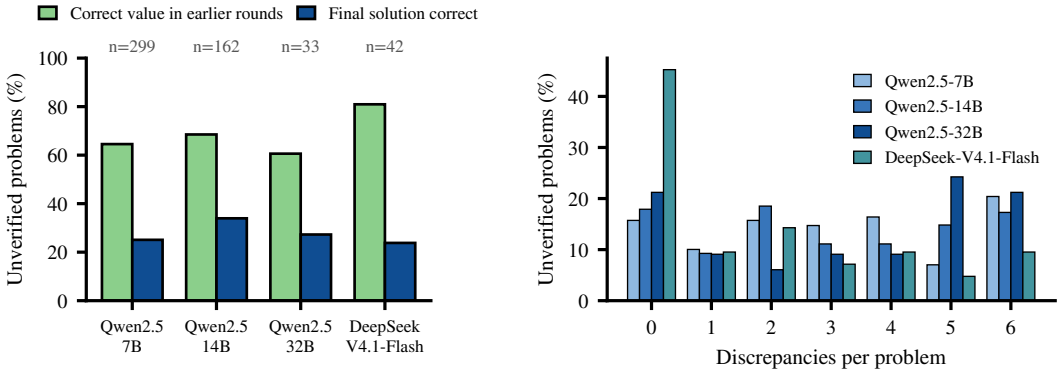

Figure 10: Unverified problems. Left: share of problems for which some earlier outcome of either stream equals the correct value, and share whose final solution is correct. n is the number of unverified problems. Right: number of discrepancies per problem, up to six.

The rounds of an unverified problem do not reach a verified value, but they often contain the correct one. For 65% of these problems with Qwen2.5-7B, 69% with Qwen2.5-14B, and 81% with DeepSeek-V4.1-Flash, some earlier outcome of the neural or the symbolic stream is correct, as the left panel of

Figure 10 shows. The short-term memory conditions the final solution on these outcomes, grouped by value, together with the discrepancies. The Qwen2.5 models leave most unverified problems with several discrepancies. DeepSeek-V4.1-Flash leaves none on 45% of them, as the right panel shows. On those problems no program variable takes different values in different rounds.

D.5 Feedback beyond the Outcome

Cross-stream verification yields feedback even when the final values of a round differ. Among rounds whose outcomes differ, the share that still contains a new lemma ranges from 11% with Qwen2.5-7B to 27% with DeepSeek-V4.1-Flash, and each such lemma enters the short-term memory. Among rounds whose outcomes match, the outcomes that the judge confirms are more accurate than those it rejects on every backbone. With DeepSeek-V4.1-Flash, confirmed outcomes are correct in 84% of the cases, against 63% for rejected ones.

E Tool Library

E.1 Tools Abstracted by the Model

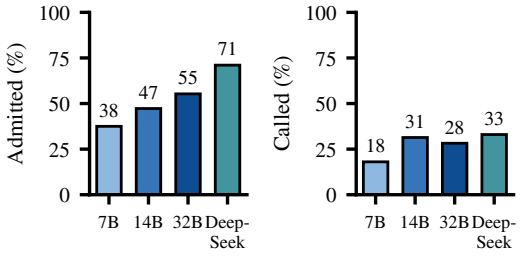

Figure 11: Tool abstraction on each backbone. Left: share of abstractions whose tool passes the unit tests of the cross-verified answers. Right: share of the problems given tools on which a program calls one.

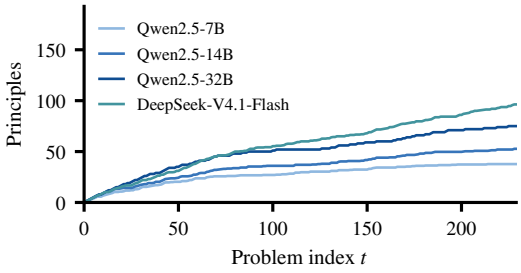

Figure 12: Principles named by the model as the problems are processed.

The tool library is the component of ChemHarness that accumulates across problems. Figure 11 traces the abstraction on every backbone, from the tools that the unit tests admit to the tools that programs call. The median admitted tool of DeepSeek-V4.1-Flash spans 34 lines of code, and a run names 96 principles. The tools compute a principle from its inputs to its result, such as the series-limit wavelength of a hydrogen spectral series, instead of a single arithmetic step. Anonymous abstractions extracted by a program are known to mislead a language model unless the model itself names and documents them (Grand et al., 2024).

The unit tests admit 38% of the abstractions of Qwen2.5-7B, 47% of Qwen2.5-14B, 55% of Qwen2.5-32B, and 71% of DeepSeek-V4.1-Flash. A program calls a selected tool on 33% of the problems given tools with DeepSeek-V4.1-Flash, against 18% with Qwen2.5-7B. Appendix E.4 shows that a library abstracted by a stronger backbone does not raise the accuracy of a weaker one.

E.2 Growth over the Problem Sequence

Figure 2 traces the tools in the library over the problem sequence, and Figure 12 traces the principles that the model names. A run of DeepSeek-V4.1-Flash names 96 principles and admits 81 tools, and the principles of every backbone keep growing over the sequence.

E.3 Rejected Abstractions

An abstraction is rejected when its rewritten programs reproduce the verified values of fewer than two source problems after the last revision. The rejections of DeepSeek-V4.1-Flash concern generalization. In 75% of its 99 rejected abstractions a program prints a value other than the verified one, and in 70%

the tool reproduces exactly one source problem. The weaker backbones fail earlier in the abstraction: the rewritten programs of Qwen2.5-7B do not call the new tool in 20% of its rejected abstractions. A tool that copies a number from the statement of a source problem accounts for 16% to 22% of the rejections.

E.4 Tool Libraries across Backbones

Table 7: Accuracy (%) with the tool library of each backbone. Each row is the backbone that solves the problems, and each column is the backbone whose frozen library it receives. Each backbone receives the libraries of stronger backbones only.

	Own library	Qwen2.5-14B	Qwen2.5-32B	DeepSeek-V4.1-Flash
Qwen2.5-7B	49.9	50.7	47.9	48.9
Qwen2.5-14B	63.3	–	64.6	63.7
Qwen2.5-32B	67.1	–	–	69.6

To separate the quality of a tool library from the backbone that uses it, a library abstracted by a stronger backbone is frozen and given to a weaker one, and Table 7 lists every pair of backbones. Qwen2.5-7B reaches 48.9% with the library of DeepSeek-V4.1-Flash and 50.7% with the library of Qwen2.5-14B, against 49.9% with its own library, and Qwen2.5-14B reaches 63.7% with the library of DeepSeek-V4.1-Flash against 63.3% with its own. A library abstracted by a stronger backbone does not move the accuracy beyond the variation between runs.

E.5 Examples of Abstracted Tools

The tools below were abstracted by DeepSeek-V4.1-Flash, and two of the four functions of each tool are listed verbatim. The first tool was abstracted from two problems of *chemmc*, and programs of two later problems called it in rounds that were verified. The second was abstracted from three problems of *matter* and *quan*.

Tool of the principle *hydrogen spectral series calculations*

```
R_H = 1.097e7  # Rydberg constant, m^-1

def hydrogen_series_limit_wavelength(n_final):
    """Compute the series-limit wavelength for a hydrogen spectral series ending at
        n_final.

    n_final: principal quantum number of the lower level (dimensionless)
    returns: series-limit wavelength in meters (float)
    """
    return 1.0 / (R_H * (1.0 / n_final**2))

def hydrogen_final_level_from_two_wavelengths(wavelength_abs, wavelength_emit,
    n_initial=1):
    """Find the final hydrogen level after absorption from n_initial then emission to a
        lower level.

    wavelength_abs: wavelength of the absorbed photon (n_initial -> m), m
    wavelength_emit: wavelength of the subsequently emitted photon (m -> n_final), m
    n_initial: starting principal quantum number before absorption, dimensionless
    returns: final principal quantum number (dimensionless)
    """
    m_sq = 1.0 / (1.0 / n_initial**2 - 1.0 / (wavelength_abs * R_H))
    m = math.sqrt(m_sq)
    nf_sq = 1.0 / (1.0 / m**2 + 1.0 / (wavelength_emit * R_H))
    return math.sqrt(nf_sq)
```

Tool of the principle *quantum particle-in-a-box*

```
H_PLANCK = 6.62607015e-34       # Planck constant, J s
C_LIGHT = 2.99792458e8          # speed of light, m/s

def box_absorption_wavelength(mass, box_length, delta_n_squared):
```

Tool of the principle *quantum particle-in-a-box* (continued)

```
    """
    Compute the photon wavelength for a particle-in-a-box transition between levels
        whose squared
    quantum numbers differ by delta_n_squared, from hc/lambda = h^2/(8 m L^2) *
        delta_n_squared.
    mass: particle mass, kg
    box_length: length of the 1D box (or cube side, using the cubic-box level spacing),
        m
    delta_n_squared: difference of squared quantum numbers n_f^2 - n_i^2 (or sum
        (n_x^2+n_y^2+n_z^2)
                     difference for a 3D cubic box), dimensionless
    returns: emitted/absorbed photon wavelength, m
    """
    return (8.0 * mass * C_LIGHT * box_length ** 2) / (H_PLANCK * delta_n_squared)

def particle_in_box_transition_quantum_number(mass, box_length, photon_wavelength):
    """
    Compute the initial quantum number n_i for a particle-in-a-box transition to the
        ground state
    emitting a photon: hc/lambda = h^2/(8 m L^2)(n_i^2 - 1), giving n_i = sqrt(8 m L^2
        c/(h lambda)+1).
    mass: particle mass, kg
    box_length: length of the 1D box, m
    photon_wavelength: wavelength of the emitted photon, m
    returns: initial quantum number n_i, dimensionless
    """
    n_squared = (8.0 * mass * box_length ** 2 * C_LIGHT) / (H_PLANCK *
        photon_wavelength) + 1.0
    return n_squared ** 0.5
```

F Error Analysis

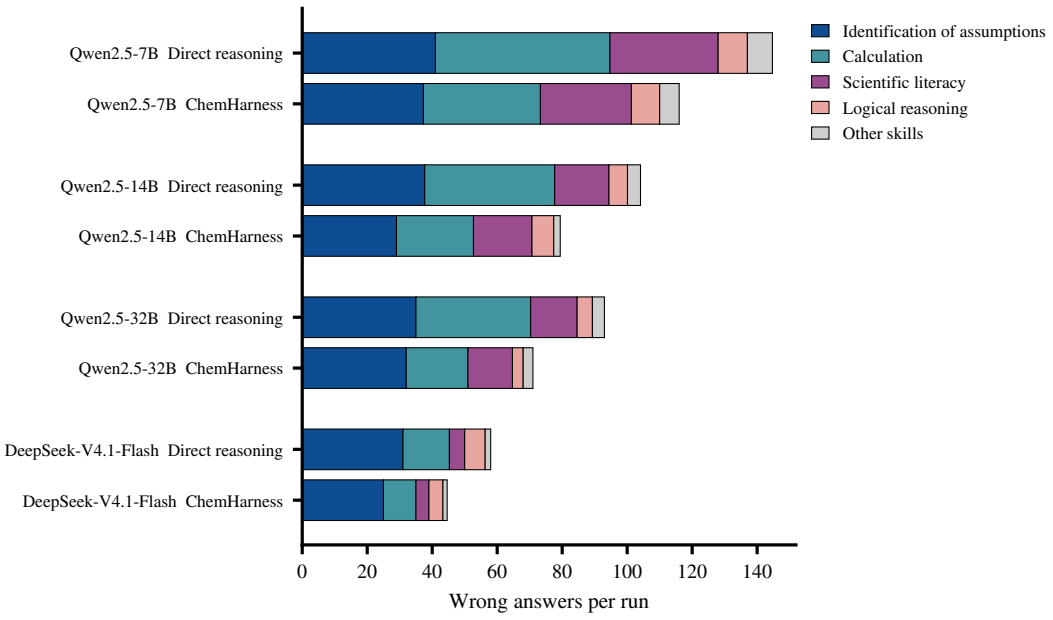

Figure 13: Wrong answers of direct reasoning and ChemHarness per run, by the skill to which the verifier attributes the error, following the error attribution of SciBench. Skills with less than 4% of all wrong answers are merged into *Other skills*.

We attribute every wrong answer to one of the ten skills of SciBench with its error attribution protocol (Wang et al., 2023b). A verifier model, GPT-5-mini, receives the problem, the correct solution, and the model's solution, and names the skill whose absence caused the error, with the system message of SciBench used verbatim. Because the test problems of SciBench lack worked solutions, the verifier receives the correct final answer and its unit in place of the correct solution. For direct reasoning

the model's solution is the first-round derivation of the neural stream, and for ChemHarness it is the solution that produces the final answer. The verifier names no skill for 25 of the 2,157 wrong answers, which are left out.

Figure 13 shows that ChemHarness produces 22% fewer wrong answers than direct reasoning over all backbones, and the reduction is largest for calculation errors, which fall by 38%. Calculation causes 36% of the errors of direct reasoning and 29% of the errors of ChemHarness. Errors in the identification of assumptions fall by 15%. Table 8 lists one example for each of the three skills to which the SciBench error attribution of Appendix F assigns most wrong answers of ChemHarness. The counts pool the 933 wrong answers of ChemHarness over all backbones.

Table 8: Examples of the three most frequent skills responsible for the wrong answers of ChemHarness with Qwen2.5-14B. The description paraphrases the verdict of the verifier.

Skill (count)	**Example**
Identification of assumptions (370 of 933)	*matter*, problem 14: the depth at which light is attenuated in seawater. The solution applies the molar absorption coefficient with an assumed absorber concentration of $1\ \mathrm{mol\ dm^{-3}}$ and answers 1.118 m against 0.87 m.
Calculation (266 of 933)	*chemmc*, problem 31: the maximum wavelength of the blackbody radiation of the earth at 288 K. The solution applies Wien's law correctly but answers 9.993×10^{-6} m against 1.01×10^{-5} m.
Scientific literacy (191 of 933)	*chemmc*, problem 13: the energy uncertainty of a γ ray from an excited state with a lifetime of 10^{-12} s. The verifier attributes the error to the form of the energy–time uncertainty relation, and the solution answers 5.273×10^{-22} J against 7×10^{-22} J.

G Case Studies

The cases below are taken from ChemHarness runs. Values, code, and memory entries are verbatim with lines omitted, and only the problem statements in Figure 14 are abbreviated. The figure shows a tool that the model abstracts from two verified solutions and admits by unit tests, a later problem whose symbolic stream calls the tool, and a short-term memory that corrects the final solution of an unverified problem.

Table 9 shows a problem that is verified in the first round. The two streams convert the pressure and the volume change separately, and the program variables that the derivation also states become lemmas. The two final values match, the judge confirms that the two solutions establish the answer, and the value of the neural stream becomes the cross-verified answer of the problem.

Table 9: A problem verified in the first round (*matter*, problem 1). Values that both streams compute are in blue.

Problem
A chemical reaction takes place in a container of cross-sectional area $50\ \mathrm{cm^2}$. As a result of the reaction, a piston is pushed out through $15\ \mathrm{cm}$ against an external pressure of $1.0\ \mathrm{atm}$. Calculate the work done by the system. Ground truth: -75 J.
Neural stream
$\Delta V = 50\ \mathrm{cm^2} \times 15\ \mathrm{cm} = 750\ \mathrm{cm^3}, \quad 750\ \mathrm{cm^3} = 750 \times 10^{-6}\ \mathrm{m^3} = 0.00075\ \mathrm{m^3}$ $1\ \mathrm{atm} = 101325\ \mathrm{Pa}, \quad W = -P\Delta V = -(101325\ \mathrm{Pa}) \times (0.00075\ \mathrm{m^3}), \quad W = -76\ \mathrm{J}$
Symbolic stream

`pressure_Pa = pressure_atm * atm_to_Pa  # 101300.0`
`delta_V = area_m2 * distance_m  # 0.00075`
`work_done_J = -pressure_Pa * delta_V  # -75.975`

Verification
Outcome level: -76.000 and -75.975 match. Step level: `pressure_Pa`, `delta_V`, and `work_done_J` are lemmas. Judge: "The written derivation and program output both provide consistent results, supporting the accuracy of the solution." Cross-verified answer: -76.000 J.

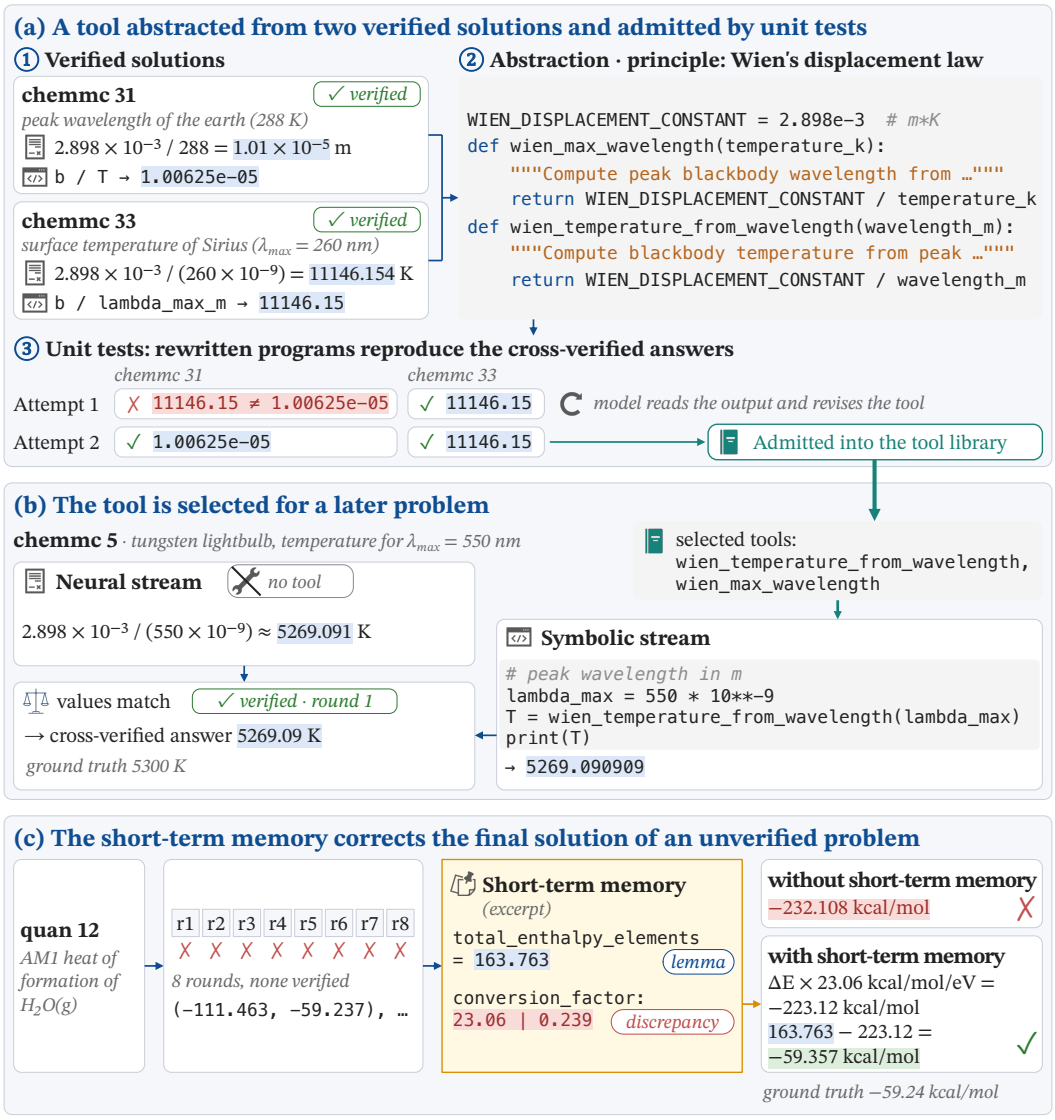

Figure 14: Examples from ChemHarness runs, with problem statements abbreviated. (a, b) DeepSeek-V4.1-Flash abstracts a tool from two verified solutions, admits it once a revised version passes the unit tests, and selects it for a later problem. (c) Qwen2.5-14B in the counterfactual replay of Section 4.7, where the same final solution is sampled with and without the short-term memory. Blue backgrounds mark values that both streams compute, and red backgrounds mark wrong values.

Table 10 lists the full problem, all eight rounds, and the memory of the unverified problem in Figure 14(c). The rounds take different values for the conversion factor from electronvolts to kilocalories per mole. The final solution conditioned on the short-term memory uses one of the two recorded factors together with a lemma and reaches the correct value, and the final solution without the short-term memory does not.

H ChemAgent Reproduction

We run the released implementation of ChemAgent with its released configuration on Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct (Tang et al., 2025). In each of three runs, the backbone builds a library from the 43 released development questions and then answers the test set, and retrieval uses

Table 10: An unverified problem (*quan*, problem 12) whose final solution is sampled with and without the short-term memory. The memory is an excerpt of the verbatim entries.

Problem
The AM1 valence electronic energies of the atoms H and O are -11.396 eV and -316.100 eV, respectively. For H_2O at its AM1-calculated equilibrium geometry, the AM1 valence electronic energy (core-core repulsion omitted) is -493.358 eV and the AM1 core-core repulsion energy is 144.796 eV. For H(g) and O(g), $\Delta H^\circ_{f,298}$ values are 52.102 and 59.559 kcal/mol, respectively. Find the AM1 prediction of $\Delta H^\circ_{f,298}$ of H_2O(g). Ground truth: -59.24 kcal/mol.
Outcomes of the eight rounds (neural, symbolic)
$(-111.463,\ -59.237)$, $(-66.404,\ 597.443)$, $(-7869.907,\ 386.758)$, $(-59.357,\ 316.286)$, $(59.257,\ -7874.077)$, $(161.931,\ 316.286)$, $(59.397,\ -3502.427)$, $(-35.993,\ 14879.939)$. No round is verified.
Short-term memory (excerpt)

`- total_enthalpy_elements = 163.763` (lemma)
`- conversion_factor: 23.06 (2 attempts, also stated by a written derivation); 0.239 (2 attempts, also stated by a written derivation)`
`- delta_H_f_H2O: 316.3 (2 attempts, also stated by a written derivation); -59.24 (1 attempt)`

Final solution without the short-term memory
$\Delta H^\circ_{f,298}(H_2O) = -8202.723$ kcal/mol $+ 163.763$ kcal/mol $= \ldots$ The answer is therefore -232.108.
Final solution conditioned on the short-term memory
$\Delta E_{\text{electronic}} \times 23.06$ kcal/mol/eV $= -9.670$ eV $\times 23.06$ kcal/mol/eV $= -223.12$ kcal/mol $\Delta H^\circ_{f,298}(H_2O(g)) = 163.763$ kcal/mol $- 223.12$ kcal/mol $= -59.357$ kcal/mol The answer is therefore -59.357.

`text-embedding-ada-002` with the released similarity threshold of 0.9. Answers are scored by the rule of Appendix B.2, and unanswered questions count as incorrect.

Table 11 summarizes the reproduction. When no library entry reaches the similarity threshold, the released code enters its imagination branch, which can end in a type error without an answer. We retain the released code, and the table lists the share of unanswered questions and the accuracy on the answered ones.

Table 11: ChemAgent reproduced with its released code and configuration, three runs per backbone. Accuracy counts unanswered questions as incorrect. Unanswered shares and library sizes are ranges over the runs.

Backbone	Accuracy (%)	Unanswered (%)	Accuracy on answered (%)	Library entries
Qwen2.5-7B	$25.5_{\pm 3.1}$	6.6–7.9	28.5	26–84
Qwen2.5-14B	$40.4_{\pm 0.9}$	15.3–16.2	48.3	88–150