In-Place Feedback: Reliable Refinement for Multi-Turn Expert-LLM Collaboration

Anonymous ACL submission

Abstract

LLM-generated drafts often contain subtle factual or logical errors, yet prior work shows that models struggle to reliably integrate multi-turn feedback aimed at fixing them. We propose *in-place feedback*, an interaction paradigm in which the user directly edits the model's previous response and the model continues generation from the edited context. In-place feedback consistently outperforms standard multi-turn feedback across five reasoning-intensive benchmarks while requiring fewer tokens, and our fine-grained analysis shows that it applies corrections more reliably and propagates them to subsequent reasoning. A user study with domain experts refining LLM-generated summaries corroborates these findings: participants report higher final-output satisfaction and substantially lower fatigue with in-place feedback, and a mixed strategy combining in-place and multi-turn feedback scores highest on every measured dimension. These results suggest that editing errors directly is a more effective paradigm for expert-LLM collaboration.

1 Introduction

Large language models (LLMs) are increasingly used as collaborative assistants in expert workflows such as software engineering, legal drafting, and clinical documentation (Lee et al., 2022; Wang et al., 2025; Zhou et al., 2024). Since expert-level tasks involve complex reasoning, mistakes can cascade and disrupt the workflow, so experts typically review outputs and provide fine-grained feedback that the model uses to produce a revised response (He et al., 2025; Chi et al., 2026; Lam et al., 2023; Satheakeerthy et al., 2025). This refinement cycle repeats until the output meets the expert's requirements, forming an expert-in-the-loop workflow.

The productivity of this workflow hinges on whether the model reliably incorporates feedback (Chidambaram et al., 2024; Vaccaro et al., 2024). Prior work shows that LLMs frequently fail to do so in multi-turn interactions (Wang et al., 2024a; Deshpande et al., 2025; Jiang et al., 2025; Laban et al., 2025), and the problem compounds as more turns accumulate: the model must integrate the latest correction while still preserving earlier feedback and leaving unrelated content untouched. When this integration fails, experts must re-check the output and re-state corrections, undermining productivity.

We hypothesize that these failures stem from the accumulation of erroneous context across turns, which obscures what the model should discard and what it should retain. To mitigate this context pollution, we propose *in-place feedback*: instead of appending a new feedback message, the user directly edits the erroneous span within the LLM's previous response and prunes the downstream portion that depends on it, and the LLM regenerates only the remainder from the corrected context. By design, in-place feedback both prevents incorrect context from accumulating and keeps the effective input shorter, reducing token usage.

We validate in-place feedback in three complementary ways.

- First, across five reasoning-intensive benchmarks spanning mathematics, science, coding, and logic (MATH-hard (Hendrycks et al., 2021), MMLU-pro (Wang et al., 2024b), GPQA (Rein et al., 2024), LiveCodeBench (Jain et al., 2025), and ZebraLogic (Lin et al., 2025)), LLMs follow in-place feedback more reliably than multi-turn feedback, with gains in both performance and token efficiency that widen across turns.

- Second, we utilize the step-by-step reasoning framework of ZebraLogic to conduct a fine-grained analysis of feedback integration.

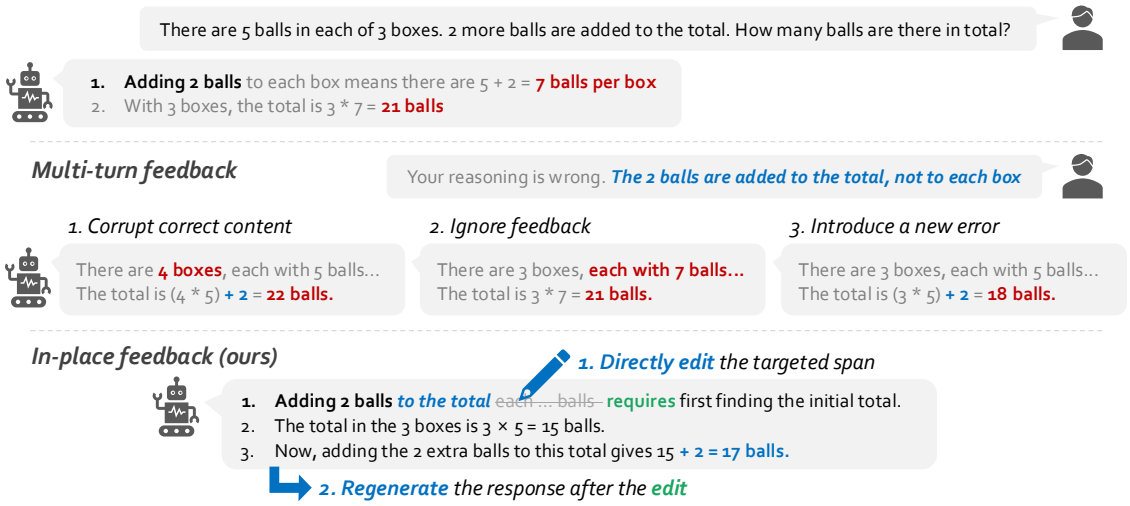

Figure 1: Illustration of common failure cases in multi-turn conversational feedback and in-place feedback. After in-place feedback, the LLM continues generation from the green word "`requires`".

Our controlled experiments reveal how standard multi-turn feedback often corrupts prior reasoning or fails to incorporate new corrections. In-place feedback mitigates these failures by regenerating directly from the context corrected by the expert, which reduces interference from stale reasoning and robustly propagates corrections.

- Third, to validate these empirical gains in a practical setting, we conduct a user study involving domain experts editing LLM-generated summaries. The participants report significantly reduced fatigue and greater satisfaction when utilizing in-place feedback. Furthermore, a hybrid workflow integrating both interaction styles achieves the highest ratings across all evaluated dimensions.

2 Related Work

Multi-turn interaction with LLMs. Two complementary directions improve how LLMs behave across turns. Architectural approaches insert clarifying questions when user input is ambiguous (Zhang and Choi, 2025; Zamani et al., 2020; Aliannejadi et al., 2019). Training-based approaches fine-tune models to better follow user intent across turns (Wu et al., 2025; Zhou et al., 2024; Shani et al., 2024; Zhang et al., 2025). Our work is orthogonal to both: we change the paradigm through which feedback enters the model, without changing the model or its training.

Refinement of LLMs. Self-refinement methods iterate a critique-and-revise loop with internal or external feedback (Madaan et al., 2023; Dhuliawala et al., 2024; Shinn et al., 2023; Nathani et al., 2023; Welleck et al., 2023). Memory-augmented variants retain feedback or summarize past turns in an external buffer (Madaan et al., 2022; Packer et al., 2023). These methods differ from ours along two axes. On the *input* side, the erroneous response is retained, so generation still conditions on incorrect content. On the *output* side, the entire response is regenerated at every turn, which inflates token cost and can overwrite spans that were already correct. In-place feedback removes the erroneous span from the conditioning context and continues generation from the corrected context.

Analysis of multi-turn LLM behavior. A growing body of work analyzes how LLMs behave across turns, reporting degraded accuracy, instruction loss, and unreliable feedback uptake (Wang et al., 2024a; Jiang et al., 2025; Laban et al., 2025; Deshpande et al., 2025). These analyses report aggregate or task-level metrics. We complement them with a turn-level decomposition (CPR, FAR, CTRR) that separates *retention* of correct content, *acceptance* of explicit feedback, and *autonomous correction* beyond stated feedback.

UI-level interactive editing. A line of HCI work proposes systems that let users directly edit problematic spans of model outputs (Masson et al., 2024; Xie et al., 2024; Pang et al., 2025). These systems offer locality at the *interface* level, but the model still conditions on the full prior response, so locality in the UI does not translate to locality in the conditioning context. We extend localized editing from the interface to the model's input, so

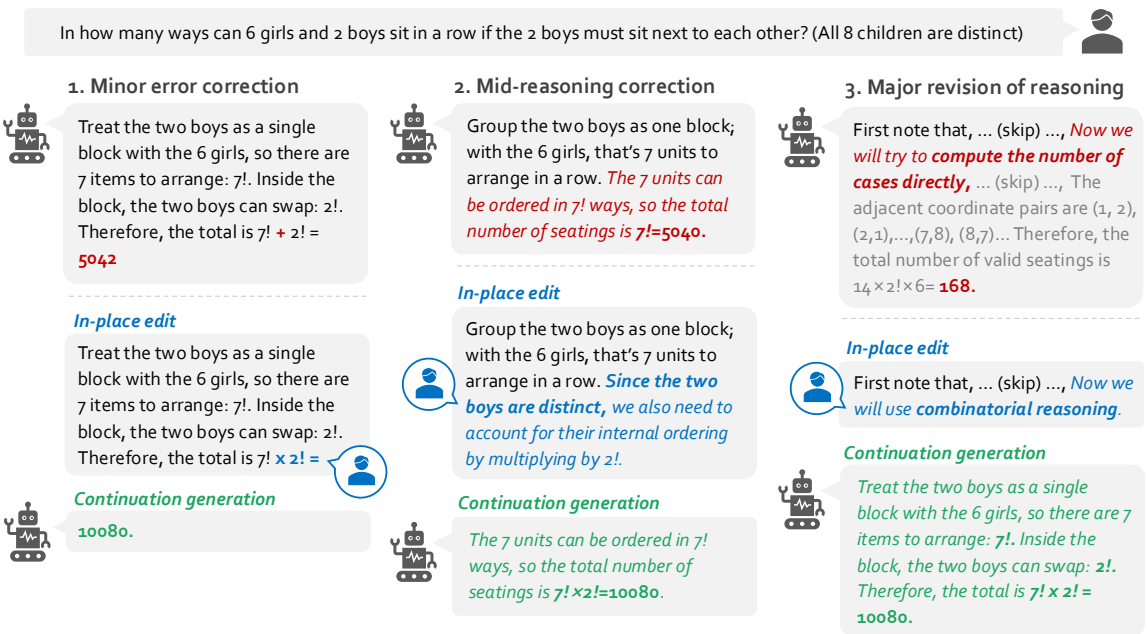

Figure 2: Representative examples of in-place feedback on a toy problem. Red marks incorrect reasoning in the original response, blue indicates user corrections with in-place feedback, and green shows the subsequent reasoning based on the corrected context. Additional examples are provided in Appendix I.1.

that the corrected span replaces the original output in the actual context used for continuation.

We provide a more detailed comparison with prior work in Appendix F.

3 In-Place Feedback

3.1 Multi-Turn Refinement with Feedback

We describe how expert feedback is incorporated into LLMs in interactive settings. Let $\mathcal{M}$ be an LLM and $\mathcal{F}$ a feedback function. Given a prompt x, the LLM produces an initial response $y_0 = \mathcal{M}(x)$, and at each subsequent turn $t \geq 1$ generates a refined response by conditioning on the prompt together with the full history of previous responses and feedback, $y_t = \mathcal{M}\left(x, y_0, f_0, \ldots, y_{t-1}, f_{t-1}\right)$, where $f_i = \mathcal{F}(x, y_i)$ denotes the feedback associated with the i-th response. We refer to this process as refinement with standard multi-turn feedback, which we hereafter simply call *multi-turn feedback*.

3.2 Motivation: Three Failure Modes of Multi-Turn Feedback

Recent work demonstrates that LLMs often fail to reliably integrate user feedback (Laban et al., 2025; Jiang et al., 2025). As illustrated in Figure 1, we identify three recurring failure modes in multi-turn refinement: (i) previously correct content becomes corrupted after receiving feedback, (ii) the model ignores the feedback and reproduces its prior mistakes, and (iii) the model incorporates the feedback locally but generates new errors in subsequent reasoning. We present real-world failure cases of multi-turn refinement in Appendix I.2.

Failures of types (i) and (ii) can substantially reduce the efficiency of expert-in-the-loop refinement by forcing experts to repeatedly re-check content or re-state feedback that is already provided. Moreover, when such failures recur, the interaction may fail to converge, meaning that sustained multi-turn feedback does not reliably yield an error-free final response. We hypothesize that these failures result from two properties of multi-turn feedback: incorrect context accumulates in the dialogue history, and feedback arrives as a separate instruction that the model should interpret and locate. While failures of type (iii) may appear to stem from the limited reasoning capability of LLMs, our analysis in Section 5 reveals that accumulated dialogue history also plays a role.

This hypothesis highlights three requirements for effective refinement. Edits from user feedback should target only the relevant reasoning step, preserve previously correct content outside the target span, and guide future reasoning from the corrected state rather than an outdated one. These considerations naturally lead us to ask: *Can we mitigate the above failures by letting the user directly edit the targeted span and constraining the model to continue generation from that point?*

3.3 In-Place Feedback

To address this question, we propose *in-place feedback*, a new interaction paradigm that treats feedback as a repair to the model's state rather than as a new instruction. As illustrated in Figure 1, our method proceeds in two stages. The first, *in-place edit*, allows the expert to directly modify the model's previous response. The expert then prunes the context that depends on the corrected span, while leaving the rest unchanged. In our setting, we assume the expert identifies one or two mistakes in the response and corrects only those parts. The second, *continuation generation*, regenerates only what is necessary to continue from the updated context. Together, these stages limit unintended changes and rebuild the response from the correction. By retaining only the corrected span and continuing from it rather than regenerating the full response, in-place feedback also keeps the dialogue history compact, an efficiency benefit we quantify in Section 4.

To illustrate how this method works in practice, Figure 2 presents representative cases of in-place feedback. In math problems, for example, in-place feedback can fix simple arithmetic mistakes or adjust flawed intermediate steps. When the error is more substantive, it can realign an incorrect reasoning path by revising larger portions of the solution.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate in-place feedback on three reasoning tasks, MATH-hard (Hendrycks et al., 2021), MMLU-pro (Wang et al., 2024b), and GPQA (Rein et al., 2024), one coding task, LiveCodeBench (Jain et al., 2025), and one logical reasoning task, ZebraLogic (Lin et al., 2025). ZebraLogic is a collection of 573 Sudoku-like puzzles in which N houses and M attributes (*e.g.*, `Name`, `Drink`, `Hobby`) form an $N \times M$ grid of cells. The values of these cells are uniquely determined by natural-language clues and uniqueness constraints. We use 500 level-5 problems from MATH-hard, the free-form subsets of MMLU-pro and GPQA (Chandak et al., 2025), and the code-generation problems from LiveCodeBench. Details are provided in Appendix A.

Evaluation. We tailor the evaluation to each task. For MATH-hard, MMLU-pro, and GPQA, since each problem has a single ground-truth answer, we evaluate a response by checking whether its final answer matches the ground-truth. For LiveCodeBench, we execute the generated code on the provided test cases and mark a response correct only if all tests pass. For ZebraLogic, each puzzle admits a unique solution that fills every cell of the grid, so we mark a response correct only when all cells match the ground-truth solution. Further details on the evaluation protocols are provided in Appendix A.

Baselines. We compare in-place feedback against two baselines: *multi-turn feedback* (Section 3.1) and *memory-based feedback*, which avoids retaining the full dialogue history by conditioning only on the latest response and the k most recent feedback items (Shinn et al., 2023; Packer et al., 2023; Madaan et al., 2022): $y_t = \mathcal{M}(P(x, y_{t-1}, f_{t-1}, \ldots, f_{t-k}))$. Here, P is a prompt template that combines the question, the previous response, and the k most recent feedback items into a single revision prompt (see Figure I.1). We set $k = 3$.

Feedback agents. To run experiments at scale without human experts in the loop, we obtain feedback from a *feedback agent* that stands in for the human experts across all methods. For benchmarks other than ZebraLogic, the agent has access to the ground-truth answer and returns a natural-language critique that points to the earliest critical error and a suggested fix, without revealing the correct final answer. We additionally post-process each critique to remove any unintended disclosure of the answer. For ZebraLogic, we instead obtain feedback systematically from a Z3 solver (De Moura and Bjørner, 2008), which identifies all conflicting cells. For each, the feedback names the cell and its correct value (*e.g.*, "`the name of House 1 is Eric, not Alice`"); we provide the top-4 most critical violations per turn by default, with a top-2 ablation in Appendix D.

In standard multi-turn feedback, the feedback is passed directly as the input for the next turn. In-place feedback, by contrast, requires the targeted error to be located within the previous response and edited in place. To automate this, in-place feedback additionally uses an *in-place agent*. The agent takes the feedback produced by the feedback agent and the model's previous response as input. It then locates the sentence targeted by the feedback and replaces it with a corrected span. Finally, it removes subsequent text that may depend on the edit.

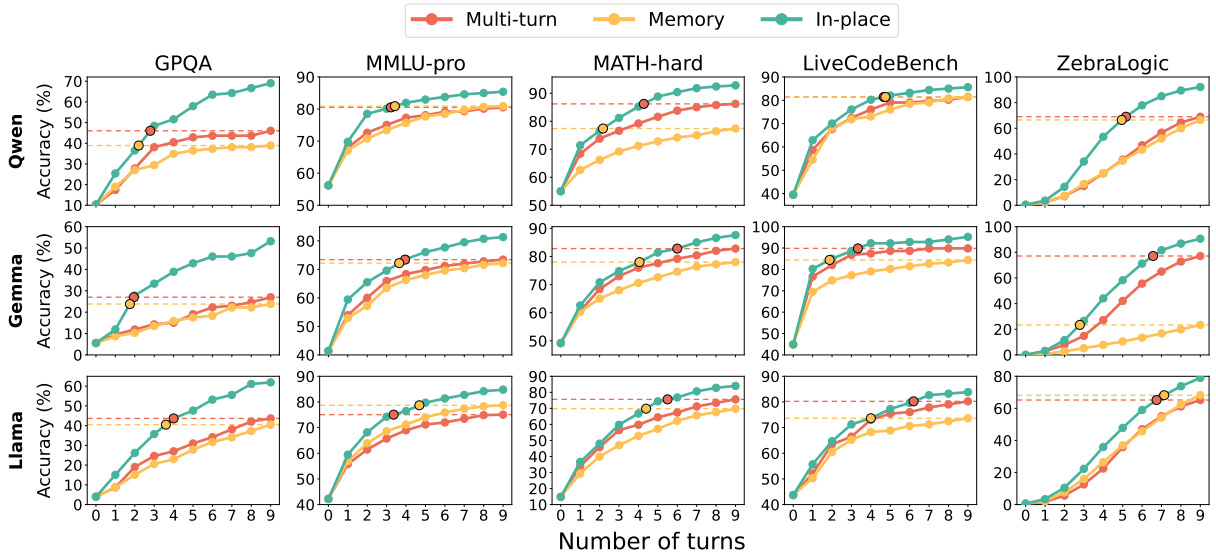

Figure 3: Performance comparison across five datasets. Across all datasets and LLMs, in-place feedback consistently outperforms the multi-turn and memory-based feedback approaches. The points with black borders represent the baselines' performance at the last turn.

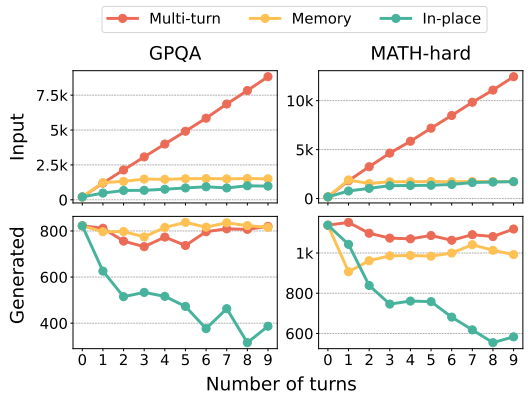

Figure 4: The number of input and generated tokens across multiple turns for `Gemma-3-4b-it`. In-place feedback consistently requires fewer tokens than multi-turn feedback. The token counts for the other models and datasets can be found in Appendix H.

We implement both agents with GPT-5-mini (OpenAI, 2025). Details such as post-processing and prompt templates are provided in Appendix A.

LLMs. We experiment with five LLMs: `Qwen2.5-7B-Instruct` (Qwen Team, 2025), `Gemma-3-4b-it` (Gemma Team, 2025), `Llama-3.1-8B-Instruct` (Kassianik et al., 2025), `Gemma-3-27b-it`, and `Llama-3.1-70B-Instruct`. The latter two are larger models that we use to examine scalability. We conduct ten refinement turns for the three smaller models and five for the two larger ones. Experimental settings are provided in Appendix A.

4.2 Results

Task performance. Figure 3 shows how accuracy changes as the number of turns increases. Across all five datasets and models, in-place feedback consistently achieves higher accuracy and exhibits faster improvement over turns. On GPQA with `Gemma`, in-place feedback improves accuracy by 47.6 percentage points over the initial response after 10 turns, an improvement more than twice that of multi-turn feedback. On MMLU-pro, its accuracy at turn 5 already surpasses the final-turn performance of multi-turn feedback across all models. Similar trends hold on ZebraLogic. These results demonstrate that in-place feedback more effectively integrates external corrections into the reasoning of LLMs. Notably, these gains hold despite the in-place agent producing appropriate edits only 40.6% of the time on MATH-hard with `Gemma` (Appendix E). The same trend holds on larger-scale LLMs (Appendix G). Qualitative examples and tabulated results are provided in Appendix I.1 and Appendix H, respectively.

In practice, feedback often only points out that a response is wrong, without indicating how to correct it. We evaluate in-place feedback under this weaker signal on ZebraLogic, replacing the feedback (*e.g.*, "`the name of House 1 is Eric, not Alice`") with its incomplete counterpart ("`the name of House 1 is not Alice`"). In-place feedback still outperforms the multi-turn baseline, since removing the erroneous span allows the model to reason from a clean state rather than anchoring to prior mistakes. Full results and discussion are

provided in Appendix E.

Token efficiency. Beyond task performance, in-place feedback is also more token-efficient. Figure 4 reports input and generated token counts. Multi-turn feedback appends each turn to the dialogue history, causing input tokens to grow linearly with turns. Both memory-based and in-place feedback avoid this accumulation and keep input length stable. The key difference emerges in generated tokens: memory-based feedback still regenerates the full response from scratch like multi-turn, whereas in-place feedback continues from the corrected span and produces substantially shorter outputs while preserving correct reasoning.

5 Failure Mode Analysis Under Controlled Experiment

Section 4 shows that in-place feedback consistently improves task performance over multi-turn feedback, but how these gains arise remains unclear. Task performance does not reveal which of the failure modes in Figure 1 are actually mitigated, specifically whether it prevents (i) corruption of previously correct content, (ii) ignoring of explicit feedback, or (iii) propagation of new errors in subsequent reasoning. It also does not show how these effects evolve as the dialogue progresses. To address this gap, we conduct a turn-level analysis on ZebraLogic, whose rule-based feedback and discrete grid structure enable us to match intermediate states against ground truth at the cell level.

5.1 Metrics for Diagnosis of Failure Modes

Each ZebraLogic puzzle is structured as a grid of cells, with one cell per entity-attribute pair. Section 4 scores a response correct only when every cell matches the solution. As a finer-grained complement, we also consider *cell accuracy*, the average proportion of cells matching the solution, with full results reported in Appendix D.1. Both, however, collapse the three failure modes from Figure 1 into a single number and do not reveal which mode dominates at which turn. To separate them, we introduce three cell-level metrics that compare consecutive responses y_t and y_{t+1}, each targeting one of the failure modes.

- **Correctness-Preserving Ratio (CPR).** The proportion of cells that are correct in y_t and remain correct in y_{t+1}, relative to the total number of cells correct in y_t. *This metric captures retention of valid reasoning and targets failure mode (i).*

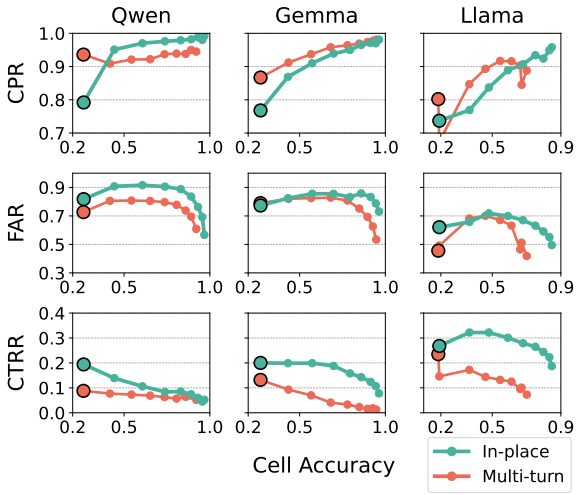

Figure 5: CPR, FAR, and CTRR for 10-turn. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

- **Feedback Acceptance Ratio (FAR).** Among cells flagged by feedback f_t, the proportion corrected in y_{t+1}. *This metric captures uptake of the explicit corrective signal and targets failure mode (ii).*

- **Correction Through Reasoning Ratio (CTRR).** Among cells incorrect in y_t but not indicated by f_t, the proportion corrected in y_{t+1}. *This metric captures whether the model can extend the correction beyond what feedback states, and targets failure mode (iii).*

5.2 Refinement Dynamics Across Turns

Observation 1. *With multi-turn feedback, models in extended dialogues reliably preserve correct content but struggle to incorporate new feedback and reason further.*

Figure 5 reports CPR, FAR, and CTRR across turns. CPR remains above 0.9 throughout, and rises with turn depth for `Gemma` and `Llama`, indicating that models retain previously correct answers more reliably as history accumulates. FAR and CTRR, however, decline steadily: FAR drops most sharply between turns 5 and 9, and CTRR eventually falls below 0.1. The models thus become less receptive to feedback and less able to reason beyond it as the dialogue lengthens. `Llama` sits at one extreme of this trade-off, attaining higher CTRR but lower CPR and FAR than the other models.

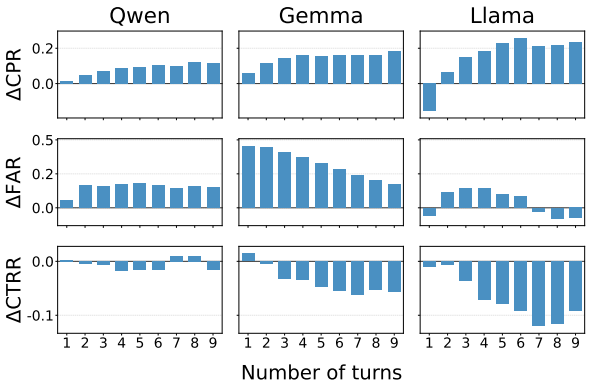

Figure 6: Impact of accumulated dialogue history. For CPR, FAR, and CTRR, we report the difference between the multi-turn feedback rates and the rates under pruned-history settings. Negative values indicate performance degradation due to accumulated history.

Observation 2. *In-place feedback is more receptive to corrections and propagates them better than multi-turn feedback.*

In-place feedback achieves a higher FAR than multi-turn feedback in later turns and consistently higher CTRR across all turns, accounting for the larger gap in accuracy. We attribute both gains to a single mechanism: the in-place edit applies the feedback directly to the response, and the model then continues from this corrected span rather than the full prior response. This reduces interference from stale reasoning and lets the model propagate the correction to cells the feedback never names explicitly. Nevertheless, even with in-place feedback, adoption remains imperfect since ZebraLogic's iterative reasoning often overwrites integrated corrections when revisiting cells.

5.3 Effect of Dialogue History for Multi-turn Feedback

While multi-turn feedback accumulates all prior interactions, in-place feedback conditions generation solely on the edited context. To isolate the effect of accumulated dialogue history, we implement an ablation: at each turn t, we re-generate the response with the context pruned to the problem, the most recent response, and the immediate feedback, $\tilde{y}_t = \mathcal{M}(P(x, y_{t-1}, f_{t-1}))$. Figure 6 reports the gap between $\tilde{y}_t$ and the original y_t across CPR, FAR, and CTRR; any difference is attributable to the presence of earlier dialogue history.

Observation 3. *While accumulating history helps models reliably reproduce correct content, it simultaneously stifles their ability to reason beyond explicit feedback.*

Accumulated dialogue history has opposing effects on retention and revision. As shown in Figure 6, the CPR gap widens with turn depth, so the full history helps the model preserve previously correct content. The CTRR gap moves in the opposite direction and turns negative for `Gemma` and `Llama`, so the same history that aids retention also obstructs the extension of reasoning beyond what feedback states. We conjecture this trade-off reflects induction-head-style copying (Olsson et al., 2022), which raises CPR by reproducing prior content but lowers CTRR by suppressing new reasoning. A deeper mechanistic analysis is left to future work. We further discuss FAR in Appendix D.3.

6 User Study

To complement the automated evaluations in Sections 4 and 5, which use LLM-based agents in place of human experts, we conduct a user study in which expert participants directly refine LLM-generated text under different feedback paradigms. The study was approved by our institutional review board.

Setup. We recruit 15 participants, each of whom has authored at least one paper in English so that they can act as domain experts on a scientific summarization task. Prior to the session, each participant nominates papers they are familiar with. During the session, the system prompts `Gemma-3-27b-it` to generate a summary of a designated portion of one such paper at a participant-specified length. The participant iteratively refines that summary across three interaction modes: (i) *multi-turn* feedback, (ii) *in-place* feedback, and (iii) a *mixed* mode in which the participant freely chooses between the two mechanisms for each correction. Each participant completes up to three topics in a 90-minute session, with the order of the three modes counterbalanced across topics to mitigate ordering effects. Procedural details and the participant interface are deferred to Appendix B and Appendix J, respectively.

Metrics. After each session, participants rate the session on a 1–10 scale along five dimensions: final-output satisfaction, feedback acceptance, correctness preservation, correction through reasoning, and fatigue. The middle three items are the human-perceived counterparts of FAR, CPR, and CTRR (Section 5), allowing us to examine whether the turn-level dynamics observed in our automated ex-

Method	Strengths	Weaknesses
Multi-turn	*"Generates natural sentences that reflect my intended nuance when I know the direction but not the exact wording.", "Good at revising the overall flow."*	*"Fatigue becomes severe when the model repeats the same mistake after correction.", "Untargeted parts get modified, and previously accepted edits are undone.", "Waiting for full regeneration every turn is tiring."*
In-place	*"I can quickly convey my intent without making the model understand it first.", "Convenient for fixing a word or phrase directly.", "Useful when the opening of a sentence is fixed and I want the model to continue naturally."*	*"The model's drafting ability is hard to leverage when I don't know how to fix it myself.", "Sentences after the edited region sometimes change slightly.", "Tedious when many small items need to be removed at once."*
Mixed	*"Restructuring with multi-turn first and then polishing details with in-place was the most efficient.", "Combines the advantages of both methods.", "Lowest fatigue and highest satisfaction overall."*	*"Choosing which mode to use for each edit is itself a small burden.", "I tended to use in-place as the default and multi-turn only as a fallback."*

Table 1: Summary of user experience across interaction modes. Representative strengths and weaknesses of three interaction modes from open-ended responses collected in our user study.

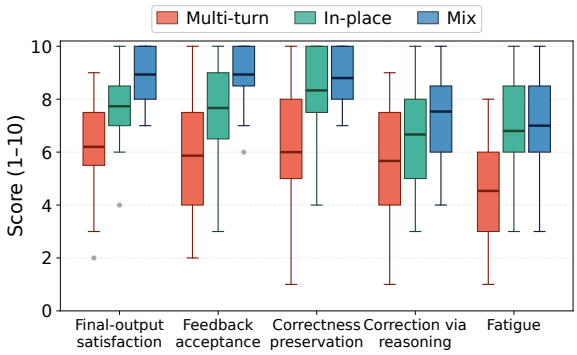

Figure 7: User study results across the five questionnaire items, scored on a 1–10 scale (higher is better; for *fatigue*, 10 corresponds to *not fatigued*). In-place feedback outperforms standard multi-turn feedback on every dimension, and a mixed strategy that combines both paradigms is rated highest overall.

periments are also reflected in expert perception. A semi-structured interview at the end of each session collects open-ended impressions; representative quotes are summarized in Section 5.3.

In-place versus multi-turn. Figure 7 shows that in-place feedback is rated above multi-turn feedback on every item, with the largest gap appearing on *fatigue*. As shown in Section 5.3, interview responses attribute the satisfaction gap to reliability rather than expressivity: participants praise multi-turn for generating natural sentences that reflect their intended nuance, but its weaknesses dominate the open-ended responses, as the model repeats already-corrected mistakes and modifies untargeted parts while undoing previously accepted edits. The fatigue gap follows from the re-verification cost imposed by full regeneration: each turn produces a fresh draft that must be re-read end-to-end, a burden amplified by the same failure modes captured by low FAR in Section 5. In-place feedback bounds this re-verification region by localizing edits and resuming generation only from the corrected span.

Mixed mode feedback. The mixed mode is rated highest on every item, and the interviews suggest that it succeeds by exploiting the complementarity of the two paradigms rather than by introducing a new mechanism. Participants describe a common workflow in which they restructure with multi-turn and then polish details with in-place, which they report as the most efficient and as yielding the lowest fatigue and highest satisfaction. The only reported downside is mild and structural: deciding which mode to use for each edit is a small burden, with several participants noting that they tend to use in-place as the default and multi-turn as a fallback.

7 Conclusion

We introduce in-place feedback, an interaction method in which the expert directly edits the model's prior response and generation continues from the edited context. Across reasoning benchmarks, controlled experiments on ZebraLogic, and a user study with domain experts refining scientific summaries, in-place feedback applies corrections more reliably across turns, lowers expert fatigue, and uses fewer tokens than standard multi-turn feedback. The user study further finds that a mixed strategy combining both paradigms scores highest on every measured dimension. We therefore recommend in-place feedback as a sensible default for expert-LLM collaboration, while exposing standard multi-turn feedback alongside it lets experts choose the most effective tool for each correction.

Limitations

Due to limited access to large-scale GPU clusters and commercial LLM APIs, we are unable to run experiments with substantially larger models or closed-source models. Moreover, in-place feedback depends on a text-continuation API, which most commercial providers do not currently offer, limiting our ability to evaluate closed-source models. As a result, our empirical evaluation focuses on models that are feasible to serve within an academic computing budget. We believe, however, that in-place feedback is not tied to a specific model size or provider, and we expect the main conclusions to continue to hold as more powerful models become available in future work.

Our automated evaluation relies on an LLM-based in-place agent to simulate expert edits at scale. As reported in Appendix E, this agent produces appropriate edits in only 40.6% of cases on MATH-Hard and 53.9% on ZebraLogic, with the dominant failure modes being incomplete repair and imperfect pruning of dependent reasoning. While in-place feedback consistently outperforms multi-turn feedback even under this noisy editor, the absolute gains reported in Sections 4 and 5 could underestimate what is achievable in real expert–LLM collaboration, where expert edits are typically cleaner and more targeted.

Our user study (Section 6) involves 15 expert participants in a scientific paper summarization task, with a 90-minute session per participant. Although the participants are domain experts who have authored English-language papers, and the three interaction modes are counterbalanced to mitigate ordering effects, broader claims about expert-LLM collaboration would benefit from larger-scale studies spanning more diverse task domains (*e.g.*, legal drafting, clinical documentation, code review), participant populations, and longer-term usage patterns that capture how preferences and fatigue evolve over repeated sessions. We leave such broader human-centered evaluation to future work.

Ethical Considerations

Our method allows users to edit an LLM response and then continue generating from that edit. However, such edits can be misused to bypass safety mechanisms. For example, a user might insert harmful instructions or unsafe text, similar to jailbreak attacks. A straightforward defense is to run user edits through a safety filter before resuming the generation process. We leave the design of defenses against such attacks to future work.

References

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W Bruce Croft. 2019. Asking clarifying questions in open-domain information-seeking conversations. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

Nikhil Chandak, Shashwat Goel, Ameya Prabhu, Moritz Hardt, and Jonas Geiping. 2025. Answer matching outperforms multiple choice for language model evaluation. *arXiv preprint*.

Wayne Chi, Valerie Chen, Ryan Shar, Aditya Mittal, Jenny Liang, Wei-Lin Chiang, Anastasios Nikolas Angelopoulos, Ion Stoica, Graham Neubig, Ameet Talwalkar, and Chris Donahue. 2026. EDIT-bench: Evaluating LLM abilities to perform real-world instructed code edits. In *The Fourteenth International Conference on Learning Representations*.

Subramanian Chidambaram, Li Erran Li, Min Bai, Xiaopeng Li, Kaixiang Lin, Xiong Zhou, and Alex C Williams. 2024. Socratic human feedback (sohf): Expert steering strategies for llm code generation. In *Findings of Empirical Methods in Natural Language Processing (EMNLP-Findings)*, pages 15491–15502.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. 2025. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of Annual Meeting of the Association for Computational Linguistics (ACL-Findings)*, pages 18632–18702.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. Chain-of-verification reduces hallucination in large language models. In *Findings of the association for computational linguistics: ACL 2024*, pages 3563–3578.

Gemma Team. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Hojae Han, Seung-won Hwang, Rajhans Samdani, and Yuxiong He. 2025. Convcodeworld: Benchmarking conversational code generation in reproducible feedback environments. In *International Conference on Learning Representations (ICLR)*.

Zhenyu He, Jun Zhang, Shengjie Luo, Jingjing Xu, Zhi Zhang, and Di He. 2025. Let the code llm edit itself when you edit the code. In *International Conference*

on Learning Representations, volume 2025, pages 59637–59653.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.

Dongwei Jiang, Alvin Zhang, Andrew Wang, Nicholas Andrews, and Daniel Khashabi. 2025. Feedback friction: Llms struggle to fully incorporate external feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Paul Kassianik, Baturay Saglam, Alexander Chen, Blaine Nelson, Anu Vellore, Massimo Aufiero, Fraser Burch, Dhruv Kedia, Avi Zohary, Sajana Weerawardhena, Aman Priyanshu, Adam Swanda, Amy Chang, Hyrum Anderson, Kojin Oshiba, Omar Santos, Yaron Singer, and Amin Karbasi. 2025. Llama-3.1-foundationai-securityllm-base-8b technical report. *Preprint*, arXiv:2504.21039.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint*.

Kwok-Yan Lam, Victor C. W. Cheng, and Zee Kin Yeong. 2023. Applying large language models for enhancing contract drafting. In *LegalAIIA@ICAIL*.

Mina Lee, Percy Liang, and Qian Yang. 2022. Coauthor: Designing a human-ai collaborative writing dataset for exploring language model capabilities. In *Proceedings of the 2022 CHI conference on human factors in computing systems*, pages 1–19.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. Zebralogic: On the scaling limits of llms for logical reasoning. In *International Conference on Machine Learning (ICML)*.

Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. 2022. Memory-assisted prompt editing to improve gpt-3 after deployment. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2833–2861.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. 2024. Directgpt: A direct manipulation interface to interact with large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, page 1–16. ACM.

Deepak Nathani, David Wang, Liangming Pan, and William Yang Wang. 2023. Maf: Multi-aspect feedback for improving reasoning in large language models. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Preprint*, arXiv:2209.11895.

OpenAI. 2025. Introducing GPT–5.

OpenAI Team. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Rock Yuren Pang, K. J. Kevin Feng, Shangbin Feng, Chu Li, Weijia Shi, Yulia Tsvetkov, Jeffrey Heer, and Katharina Reinecke. 2025. Interactive reasoning: Visualizing and controlling chain-of-thought reasoning in large language models. *Preprint*, arXiv:2506.23678.

Qwen Team. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *Conference on Language Modeling (COLM)*.

Shrirajh Satheakeerthy, Daniel Jesudason, James Pietris, Stephen Bacchi, and Weng Onn Chan. 2025. Llm-assisted medical documentation: efficacy, errors,

and ethical considerations in ophthalmology. *Eye*, 39(8):1440–1442.

Lior Shani, Aviv Rosenberg, Asaf Cassel, Oran Lang, Daniele Calandriello, Avital Zipori, Hila Noga, Orgad Keller, Bilal Piot, Idan Szpektor, et al. 2024. Multi-turn reinforcement learning with preference human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Michelle Vaccaro, Abdullah Almaatouq, and Thomas Malone. 2024. When combinations of humans and ai are useful: A systematic review and meta-analysis. *Nature Human Behaviour*, 8(12):2293–2303.

Jian Wang, Yinpei Dai, Yichi Zhang, Ziqiao Ma, Wenjie Li, and Joyce Chai. 2025. Training turn-by-turn verifiers for dialogue tutoring agents: The curious case of llms as your coding tutors. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12416–12436.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024a. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations (ICLR)*, volume 2024, pages 32593–32627.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. 2024b. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *International Conference on Learning Representations (ICLR)*.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. 2025. Collabllm: From passive responders to active collaborators. In *International Conference on Machine Learning (ICML)*.

Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. Waitgpt: Monitoring and steering conversational llm agent in data analysis with on-the-fly code visualization. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, page 1–14. ACM.

Hamed Zamani, Susan Dumais, Nick Craswell, Paul Bennett, and Gord Lueck. 2020. Generating clarifying questions for information retrieval. In *Proceedings of The Web Conference (WWW)*.

Michael JQ Zhang and Eunsol Choi. 2025. Clarify when necessary: Resolving ambiguity through interaction with lms. In *Findings of Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-Findings)*.

Michael JQ Zhang, W Bradley Knox, and Eunsol Choi. 2025. Modeling future conversation turns to teach llms to ask clarifying questions. In *International Conference on Learning Representations (ICLR)*.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024. Archer: Training language model agents via hierarchical multi-turn rl. In *International Conference on Machine Learning (ICML)*.

Appendix

A Experimental Details

A.1 Datasets

MMLU-pro and GPQA free-form. The MMLU-pro and GPQA-Diamond datasets are employed to evaluate the knowledge-intensive reasoning abilities of LLMs in free-form question answering (Chandak et al., 2025). Both datasets originate from multiple-choice benchmarks but are adapted for generative evaluation via answer matching.

Motivation for free-form evaluation. Multiple-choice evaluation has intrinsic limitations: models can exploit statistical patterns in the answer choices without engaging in critical reasoning. As a result, multiple-choice accuracy overestimates the model's ability to generate correct answers. When the choices are removed and free-form responses are required, models are forced to generate an answer directly, aligning the evaluation more closely with the capabilities that matter in real-world use cases.

Selection of questions. MMLU-pro is derived from the MMLU benchmark and initially includes approximately 12,000 questions across various domains. To ensure that the questions are answerable without the provided answer choices, an automatic filtering process is employed. This process uses a rubric-based grader to narrow down the dataset to about 5,500 questions. From this reduced pool, questions that are sufficiently specific and have a unique correct answer are selected. After this, a manual filtering step yields a final set of 493 questions. This dataset offers comprehensive coverage across diverse domains, making it a useful resource for evaluating general knowledge and reasoning skills.

The GPQA-Diamond dataset comprises 198 graduate-level science questions, designed to be challenging and to test in-depth knowledge and critical reasoning. As with MMLU-pro, a filtering process is applied to select questions with clear and specific correct answers, resulting in a final set of 126 questions. This dataset is more focused and rigorous, emphasizing high-quality scientific questions that demand deep reasoning.

These preparation steps yield two complementary free-form datasets: MMLU-pro, which provides broader coverage across domains with 493 carefully filtered and annotated items, and GPQA-Diamond, which offers a smaller but more rigorous collection of 126 scientific questions. This ensures that free-form evaluations are conducted only on questions with clear, unambiguous solutions.

Evaluation. We evaluate model answers in two stages on MATH-hard, MMLU-pro, and GPQA. We first attempt exact matching. If it fails, we then apply an LLM judge to identify semantically equivalent answers expressed in different forms. In LiveCodeBench, we evaluate correctness by executing the provided test cases. We use GPT-oss-20b (OpenAI Team, 2025) as the judge model.

A.2 ZebraLogic

The ZebraLogic dataset comprises logic grid puzzles designed to assess the reasoning capabilities of LLMs. Each puzzle is structured around a grid with a certain number of houses and attributes. Specifically, each puzzle involves N houses and M attributes, creating an $N \times M$ grid that needs to be filled. The attributes in these puzzles are distinct for each house, where each attribute has N unique values corresponding to the houses.

The puzzles come with a set of clues that impose logical constraints on the grid. For example, one clue might specify that "`The person who likes milk is Eric`", while another might state, "`The person who drinks water is Arnold`". These clues help guide the reasoning process to fill in the grid, ensuring that all constraints are satisfied. Importantly, each puzzle has a unique solution, which guarantees that any feedback provided for solving the puzzle is definitive and accurate. This structure ensures that ZebraLogic provides a rigorous framework for evaluating logical reasoning in models, where the set of clues provided uniquely determines each puzzle's solution.

Puzzle generation. Puzzles are generated by first sampling a complete solution, then constructing a superset of consistent clues from a fixed inventory. The clue types are as follows: `FOUNDAT`, `SAMEHOUSE`, `NOTAT`, `DIRECTLEFT/RIGHT`, `SIDEBYSIDE`, `LEFT/RIGHTOF`, and `ONE/TWOBETWEEN`. Each clue type provides a constraint by capturing a specific relationship between variables. A minimal subset of clues is retained through iterative pruning while preserving the uniqueness of the solution. This guarantees that puzzles are neither under-specified nor trivially over-constrained.

Dataset filtering. The ZebraLogic dataset contains 1,000 puzzles spanning all combinations of $N, M \in \{2, \ldots, 6\}$. We filter the dataset by puzzle difficulty based on the search space size, which is defined as the total number of possible configurations that satisfy only the uniqueness constraints of the puzzle; for a puzzle with N houses and M attributes, the search space size is $|\mathcal{S}| = (N!)^M$. Puzzles with small search spaces ($|\mathcal{S}| < 10^3$) are excluded, along with those of size 3×4 and 3×5, as well as invalid puzzles (*e.g.*, cases where distinct categories share identical attribute values). This yields a controlled, reasoning-intensive testbed for analyzing the feedback acceptance of LLMs in multi-turn settings.

Input and output format. Puzzles are presented in natural language, followed by an instruction asking the model to fill the $N \times M$ grid. The input template prompt is provided in Figure I.6. The expected output is a structured JSON table. This format enables automatic cell-level evaluation and the application of fine-grained feedback. We attempt up to 30 re-generations to obtain a syntactically valid JSON output. If all attempts fail, we treat the instance as wrong and omit it from CPR, FAR, and CTRR calculations.

For JSON-parsed predictions, we employ a fuzz score from the Python rapidfuzz library to perform approximate string matching. Specifically, we compute the highest similarity score among the candidate attributes, and if the score exceeds 50, we adopt the corresponding attribute as the predicted value.

Feedback functions. We generate rule-based feedback using a fixed template, as illustrated in Figure I.7. For `Llama`, we additionally append reasoning guidance to the feedback, since the model frequently produces only the final JSON-formatted answer without including the corrected reasoning process.

In-place agent. The in-place agent simulates a human editor by directly modifying the LLM's response during evaluation. In math problems, each reasoning step depends on the previous one, so a correction usually requires discarding subsequent steps. Zebra puzzles, however, follow a different structure. They involve parallel reasoning, in which multiple constraints must be satisfied simultaneously. As a result, later reasoning steps can remain valid even if earlier ones are incorrect. To handle this, we retain subsequent reasoning steps after in-place feedback is applied.

We first segment the model's response into reasoning steps. The agent checks each step against the received feedback and edits the response when a directly mispredicted attribute value is specified (*e.g.*, changing `Name of House 2 = Alice` to `Name of House 2 = Eric`). If there are reasoning steps that depend on the mispredicted value identified by the feedback (*e.g.*, reasoning built on `Alice in House 2`), those dependent steps are removed to prevent error propagation. After applying these edits or deletions, the final solution is removed, and the prompt `Further reasoning:` is appended to encourage continuation of the reasoning process. We employ GPT-5-mini as the in-place feedback agent, following the rule prompt in Figure I.8.

A.3 Prompt and Post-Processing

Empirical experiments. We define answer leakage as any explicit revelation of the ground-truth answer within feedback or intervention outputs. Following Jiang et al. (2025), we prevent it by using the prompts in Figures I.3 to I.5 and applying post-processing for each agent: (i) after generating feedback, we scan the message and mask any span that reveals the ground truth; (ii) for the in-place feedback agent, if any part of the message exposes the solution, we prune those spans before presenting the message to the model.

ZebraLogic. We constrain the in-place feedback agent to avoid introducing reasoning beyond the scope of the provided feedback. The model's output is segmented into discrete reasoning steps, which are then checked for consistency with the provided feedback. In cases of conflict, the corresponding step is minimally revised, following the prompt in Figure I.8. Notably, the agent does not have direct access to the puzzle itself, which limits its ability to extend reasoning beyond the explicitly given feedback.

A.4 LLMs.

We experiment with five LLMs: `Qwen2.5-7B-Instruct` (Qwen Team, 2025), `Gemma-3-4b-it` (Gemma Team, 2025), `Llama-3.1-8B-Instruct` (Kassianik et al., 2025), `Gemma-3-27b-it`, and `Llama-3.1-70B-Instruct`. The latter two are larger models that we use to examine scalability. We use publicly available models released under

their respective licenses.

A.5 Hyperparameters

All models use temperature 0. We observed that the `Llama` family can be overly verbose, resulting in degradation of generation quality. To stabilize decoding, we set its repetition penalty to 1.15. All other models use a repetition penalty of 1.0. The maximum generation length is 2048 tokens for all models and experiments. We use vLLM (Kwon et al., 2023) for efficient inference.

A.6 Memory-based Feedback

Following Shinn et al. (2023), we retain at most the three most recent feedback items. Figure I.1 shows the prompt template for memory-based feedback.

B User Study Details

B.1 Study Setup

Participants. We recruit 15 participants. To qualify, each participant must have authored at least one paper written in English. This criterion ensures that participants have the reading comprehension and academic writing skills required to act as domain experts: they can reliably judge whether a model-generated summary faithfully reflects a source paper and can issue targeted, well-formed corrections. All participants provide informed consent, take part voluntarily, and may withdraw at any time. Each participant receives $20 as compensation for the 90-minute session. The study was approved by our institutional review board. All participant data is anonymized prior to analysis.

Task. We conduct a user study on a scientific summarization task. Prior to the session, each participant nominates papers they are already familiar with. At the start of each topic, the participant specifies the desired summary length, which is then incorporated into the system prompt provided to the LLM. The system subsequently prompts the LLM to generate a summary of the designated portion of the paper at the requested length. Since the participant knows the source material well, they can refine the generated summary, not only correcting factual or logical errors but also improving its wording, structure, and completeness, mirroring the expert-in-the-loop refinement setting. We place no restriction on the kinds of edits a participant may make, though we recommend against vague, sweeping revisions (*e.g.*, paraphrasing the entire summary at once). Within a 90-minute session, each participant completes up to three summarization topics, each corresponding to a distinct paper or section of a paper.

Interaction modes. We compare three interaction modes: (i) *standard multi-turn feedback*, where the participant appends natural-language feedback messages and the model regenerates the full response; (ii) *in-place feedback*, where the participant directly edits the erroneous span and the model continues generation from the edited context; and (iii) *mixed mode*, where the participant may freely choose, for each correction, between the two mechanisms. For each summarization topic, the participant carries out the refinement process under all three interaction modes, so that the conditions can be compared on the same source material. The order of the three modes is counterbalanced across topics and participants to mitigate ordering effects.

Model. We use `Gemma-3-27b-it` as the underlying LLM for all conditions with a temperature of 0.0, consistent with the large-scale model setting in our main experiments.

Procedure. Each session begins with a short onboarding in which participants are introduced to the interface and the three feedback mechanisms, followed by a practice trial on a warm-up paper to familiarize them with the workflow. Participants then complete their summarization topics, each under all three interaction modes. We provide examples of the user study interface in Appendix J. After the session, participants complete a post-study survey to evaluate each of the three feedback mechanisms.

Metrics. After each interaction mode, participants answer five questions, each rated on a 1–10 scale:

- **Final-output satisfaction.** How satisfied the participant is with the final generated summary produced under this method.
- **Feedback acceptance.** Whether the next-turn response sufficiently incorporates the feedback provided by the participant.
- **Correctness preservation.** Whether the next-turn response preserves the parts the participant is already satisfied with and does not want changed.

- **Correction through reasoning.** Whether the next-turn response also satisfactorily improves parts that are not explicitly mentioned in the participant's feedback.

- **Fatigue.** The participant's overall fatigue while refining the output with this mode, taking into account all relevant factors such as task efficiency and the difficulty of the feedback mechanism.

The feedback acceptance, correctness preservation, and correction through reasoning items are the human-perceived counterparts of the Feedback Acceptance Ratio (FAR), Correctness-Preserving Ratio (CPR), and Correction Through Reasoning Ratio (CTRR) defined in Section 5, respectively, allowing us to examine whether the turn-level dynamics observed in our automated experiments are also reflected in expert perception. In addition to the questionnaire, we conduct a semi-structured interview at the end of the session to collect open-ended impressions of each feedback paradigm, including perceived control, effort, and trust.

B.2 Representative User Comments

Section 5.3 summarizes representative strengths and weaknesses of each interaction mode, drawn from the open-ended responses collected in our post-study survey. Non-English comments were translated into English using an LLM, and all resulting entries were manually reviewed and verified by the authors to ensure that the original intent and nuance were faithfully preserved.

C The Use of Large Language Models

We used an LLM assistant as a writing tool for grammar checking and paraphrasing. In addition, since our experiments required carefully designed prompts, we employed the assistant to refine prompts.

D Extended ZebraLogic Results

D.1 Grid Accuracy and Cell Accuracy

Figures D.1 and D.2 show performance of methods on grid show grid accuracy and cell accuracy, respectively. Memory-based feedback performs the worst among the three methods. Figures D.4 and D.5 show that, compared with multi-turn feedback, memory-based feedback has smaller declines in FAR and CTRR as turns evolve. This pattern may partly reflect its lower overall performance. However, our controlled experiments in Section 5.3 further indicate that the accumulation of dialogue history can contribute to the decreases in FAR and CTRR observed under multi-turn feedback.

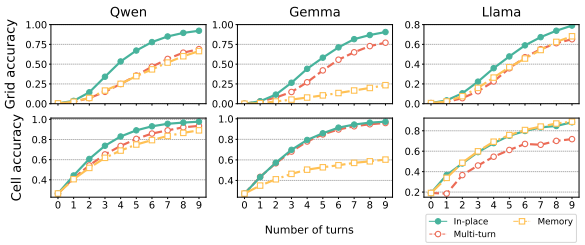

Figure D.1: Grid and cell accuracy on the ZebraLogic dataset. Top-4 feedback is used.

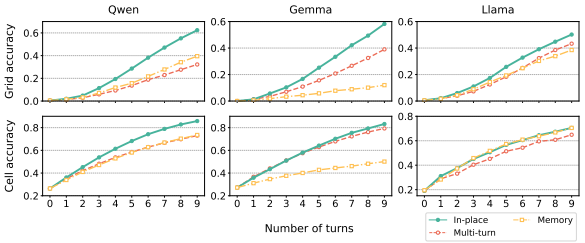

Figure D.2: Grid and cell accuracy on the ZebraLogic dataset. Top-2 feedback is used.

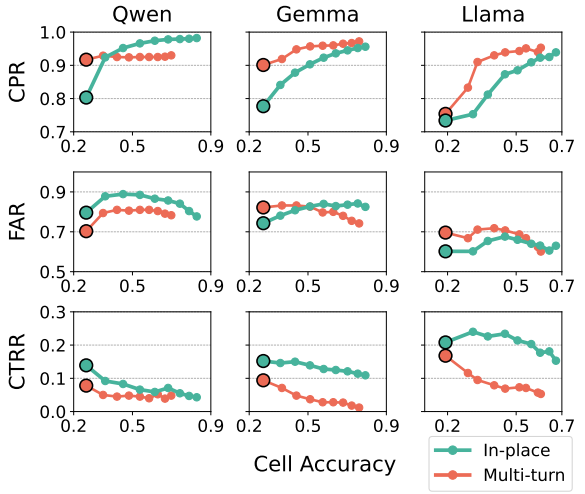

Figure D.3: CPR, FAR, and CTRR for 10-turn with top-2 feedback. Filled points denote in-place feedback, and hollow points denote multi-turn feedback. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

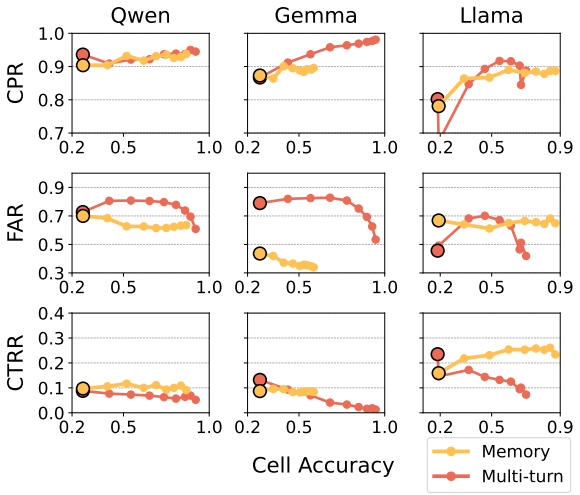

Figure D.4: CPR, FAR, and CTRR for 10-turn conversations of LLMs on the ZebraLogic dataset. Top-4 feedback is used. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

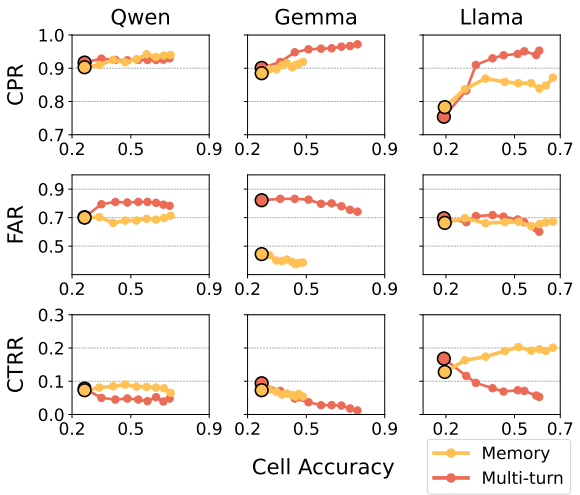

Figure D.5: CPR, FAR, and CTRR for 10-turn conversations of LLMs on the ZebraLogic dataset. Top-2 feedback is used. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

D.2 Performance under Oracle Feedback

In addition to top-k feedback, we conduct an experiment with *oracle feedback*, in which feedback is provided for every incorrect cell rather than only the top-k most critical ones. Figure D.6 presents grid and cell accuracy as a function of the number of turns. In-place feedback converges substantially faster than multi-turn feedback, indicating that it enables the model to incorporate feedback more efficiently. Figure D.7 illustrates the turn-level behavior of LLMs under oracle feedback. In most cases, in-place feedback achieves higher CPR and FAR than multi-turn feedback; the only exception is the CPR of `Qwen` in the first turn.

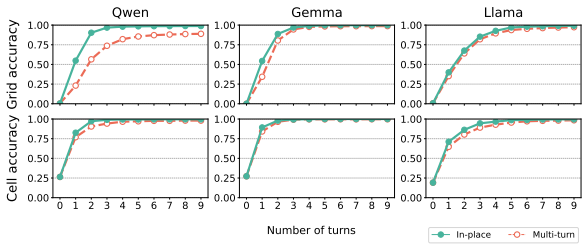

Figure D.6: Grid and cell accuracy of LLMs on the ZebraLogic dataset. In-place feedback consistently outperforms multi-turn feedback under an oracle setting.

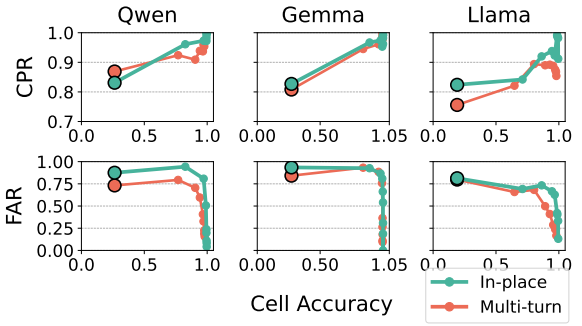

Figure D.7: CPR, FAR, and CTRR for 10-turn conversations of LLMs on the ZebraLogic dataset. The oracle feedback function is used. Filled points denote in-place feedback, and hollow points denote multi-turn feedback. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

D.3 Extended Analysis on Dialog History

We extend the discussion in Section 5.3.

Observation 4. *FAR trends under multi-turn feedback vary across models, unlike the uniform trends observed in CPR and CTRR.*

`Qwen` maintains consistently high FAR, `Gemma` shows diminishing gains in later turns, and `Llama` benefits initially but reverses in later turns. Since FAR is closely tied to instruction-following capabilities, which are heavily shaped by model-specific post-training, we attribute these patterns to a complex interplay between post-training differences and shared mechanistic inductive biases.

In-place feedback sidesteps this heterogeneity, reaching the highest FAR across all three models in later turns (Figure 5). We attribute this to its

mechanism: in-place feedback places the correction directly into the response, rather than sending it as a separate instruction for the model to interpret and apply. This removes the need to treat feedback as an external command, yielding consistently high FAR regardless of model-specific instruction-following behavior.

E Robustness to Imperfect Feedback

To assess the robustness of in-place feedback under realistic conditions, we conduct two complementary analyses of the quality of (i) the feedback content and (ii) the editing process.

E.1 Incomplete Feedback

We design a controlled experiment on ZebraLogic in which the feedback identifies the erroneous cell but does not reveal the correct value. Specifically, the standard top-k feedback template *"⟨category⟩ of ⟨house⟩ is ⟨gt⟩, not ⟨pred⟩"* is replaced with *"⟨category⟩ of ⟨house⟩ is not ⟨pred⟩"*. We provide the top-4 incomplete feedback signals per instance and run 10 turns with three open-source LLMs.

Appendix E.1 report grid and cell accuracy. Although absolute performance is substantially lower than under complete feedback since the model must independently derive the correct value, in-place feedback consistently outperforms multi-turn feedback across all three models and both metrics. We attribute this robustness to the fact that in-place feedback removes erroneous spans from the context, letting the model focus on the localized correction without being misled by previously incorrect content. Multi-turn feedback, in contrast, retains the full erroneous output in the conversation history, which can interfere with reasoning when the model must derive the correct value on its own.

This setting is particularly demanding since the model cannot simply copy the correct value from the feedback; it must reason from scratch, given only the negative signal. The fact that in-place feedback retains its advantage in this regime indicates that the benefit does not stem from the explicitness of the corrective content alone, but from the cleaner reasoning context that direct editing affords.

E.2 Reliability of the In-place Agent

A second source of imperfection is the in-place agent itself, which must localize the conflicting span, apply a minimal edit, and prune dependent reasoning. To analyze its reliability, we classify edits from MATH-Hard and ZebraLogic (top-4 feedback) with Gemma-3-4b-it, using Claude 4.5 Sonnet as an LLM-judge according to the taxonomy in Appendix E.2.

Model	Method	0	1	2	3	4	5	6	7	8	9
Qwen	Multi-turn	0.005	0.009	0.010	0.014	0.016	0.017	0.017	0.019	0.021	0.024
	In-place	0.005	0.009	**0.017**	**0.028**	**0.037**	**0.049**	**0.059**	**0.068**	**0.079**	**0.086**
Gemma	Multi-turn	0.000	0.002	0.003	0.005	0.009	0.010	0.017	0.019	0.021	0.023
	In-place	0.000	**0.009**	**0.018**	**0.034**	**0.042**	**0.044**	**0.055**	**0.067**	**0.076**	**0.090**
Llama	Multi-turn	0.007	0.014	0.017	0.019	0.021	0.026	0.031	0.033	0.037	0.038
	In-place	0.007	**0.032**	**0.043**	**0.053**	**0.059**	**0.069**	**0.085**	**0.090**	**0.106**	**0.106**

Table E.1: Grid accuracy on ZebraLogic under *incomplete* top-4 feedback. Feedback indicates only that a cell is incorrect, without providing the correct value.

Model	Method	0	1	2	3	4	5	6	7	8	9
Qwen	Multi-turn	0.274	0.284	0.290	0.296	0.295	0.297	0.296	0.299	0.300	0.299
	In-place	0.274	**0.304**	**0.341**	**0.372**	**0.395**	**0.414**	**0.434**	**0.444**	**0.453**	**0.449**
Gemma	Multi-turn	0.273	0.283	0.307	0.319	0.333	0.344	0.346	0.346	0.354	0.361
	In-place	0.273	**0.309**	**0.336**	**0.371**	**0.392**	**0.414**	**0.424**	**0.449**	**0.461**	**0.472**
Llama	Multi-turn	0.188	**0.208**	0.218	0.234	0.238	0.254	0.292	0.312	0.324	0.338
	In-place	0.188	0.169	**0.315**	**0.358**	**0.394**	**0.416**	**0.438**	**0.453**	**0.454**	**0.457**

Table E.2: Cell accuracy on ZebraLogic under *incomplete* top-4 feedback.

The agent produces appropriate edits in only 40.6% (MATH-Hard) and 53.9% (ZebraLogic) of cases. The dominant failure mode is incomplete repair: on MATH-Hard, the agent often partially addresses the feedback (33.4%) or over-prunes (18.2%); on ZebraLogic, under-pruning is most common (39.9%), reflecting the parallel reasoning structure in which dependencies between cells are harder to trace. Excessive rewriting is rare ($\leq$1.6%) on both datasets. This is the failure mode most likely to inflate performance through unintended content changes, so its rarity suggests that the gains reported in Sections 4 and 5 are not artifacts of the agent rewriting beyond the targeted span.

Takeaway. Together, these analyses show that in-place feedback is robust to imperfection from *both* the feedback content (incomplete signals) and the editing process (imperfect agent edits). Both factors degrade in-place feedback, yet it consistently outperforms multi-turn feedback. We therefore expect the gap to be *larger* in real expert-LLM collaboration, where human edits are typically cleaner and more targeted than those produced by an automatic agent.

F Extended Related Work

Multi-turn interaction of LLMs. Several studies aim to improve the performance of LLMs in multi-turn interaction. One line of work focuses on clarifying questions, where the LLM generates follow-up questions when the user's input is am-

Category	Sub-type	MATH (%)	Zebra (%)	Description
Appropriate	–	40.6	53.9	Correct localization, minimal fix, proper pruning.
Incomplete	Over-prune	18.2	0.7	Valid reasoning discarded beyond what feedback necessitated.
	Under-prune	3.8	39.9	Erroneous downstream content retained.
	Partial address	33.4	–	Feedback partially incorporated.
	Misapplied	2.4	4.2	Feedback misunderstood or not reflected in the edit.
Excessive	–	1.6	1.3	Rewriting valid content beyond feedback scope.

Table E.3: Reliability analysis of the in-place agent on MATH-Hard (Gemma-3-4b-it) and ZebraLogic (top-4 feedback), evaluated by Claude 4.5 Sonnet as LLM-judge. Most edits are appropriate or incomplete; aggressive over-rewriting (excessive) is rare.

biguous (Zhang and Choi, 2025; Zamani et al., 2020; Aliannejadi et al., 2019). Zhang and Choi (2025) proposes a framework that integrates clarifying questions into the response generation process. Another line of work enhances multi-turn performance through additional training, often employing a user simulation model to generate multi-turn interaction scenarios (Zhou et al., 2024; Shani et al., 2024; Zhang et al., 2025; Wu et al., 2025). For example, Wu et al. (2025) fine-tunes LLMs with reinforcement learning to improve user interactivity. Our work focuses on how LLMs can achieve more effective interaction with users without additional training, while ensuring token efficiency.

Refinement of LLMs. Recent research explores self-refinement, an approach in which LLMs generate feedback on their own outputs and improve them accordingly (Madaan et al., 2023; Dhuliawala et al., 2024; Shinn et al., 2023; Nathani et al., 2023). Self-Refine (Madaan et al., 2023), for instance, iterates a critique-then-revise loop in which the model regenerates the full response at each step while previous responses remain in the dialogue history. Huang et al. (2024) shows that LLMs often fail to self-correct reasoning errors without external signals. In response, Welleck et al. (2023) trains a separate model to produce feedback, and Han et al. (2025) uses an external LLM agent to provide feedback for evaluating model performance. A complementary line of work modifies the prompt or external memory based on user feedback to avoid recurring mistakes. MemPrompt (Madaan et al., 2022) stores user clarifications in a memory and retrieves them to rewrite future prompts, and MemGPT (Packer et al., 2023) selectively manages context by summarizing or discarding older interactions to fit a bounded working memory. These approaches differ from ours along two axes. First, on the input side, the erroneous response itself is retained, either verbatim in the dialogue history (Self-Refine) or in summarized form in memory (MemPrompt, MemGPT), so the model still conditions on incorrect outputs when producing its next response. Second, on the output side, all of these methods regenerate the entire response in every iteration, which is token-inefficient and can introduce new errors in spans that were already correct. In contrast, our method removes the erroneous span from the conditioning context and generates only the continuation after the corrected point, so that both the conditioning input and the regenerated output are restricted to the region affected by the error.

Analysis of multi-turn conversations. Recent studies analyze the performance of LLMs in multi-turn conversations. Jiang et al. (2025) shows that LLMs fail to reliably incorporate feedback, even when it is close to the correct answer, highlighting the need for stronger correction mechanisms. Laban et al. (2025) finds that accuracy decreases when a single problem is divided into multiple parts and solved by LLMs in a multi-turn manner. Deshpande et al. (2025) analyzes the performance of LLMs in multi-turn conversations across four categories, including instruction retention and self-coherence. While these studies analyze the performance of LLMs in the multi-turn setting, they primarily report high-level metrics, such as overall task accuracy, without analyzing how reasoning evolves across turns or how errors propagate. Our turn-level diagnostic metrics (CPR, FAR, CTRR) directly address this gap by quantifying when and how erroneous context degrades subsequent reasoning.

UI-level interaction systems. Prior HCI research has investigated UI-level systems that provide localized, in-place interventions when interacting with LLMs (Pang et al., 2025; Masson et al., 2024; Xie

et al., 2024). These systems allow users to directly edit problematic spans or to monitor intermediate reasoning steps and execution traces to diagnose and guide multi-step behavior. This line of work shows that UI-level support for localized, in-place editing helps users work more efficiently and with lower cognitive load, since they can precisely identify and correct errors. While these systems offer locality at the interaction level, at the implementation level, generation is still conditioned on the entire prior response, so locality in the UI does not translate into locality in the conditioning context. Motivated by these findings, we extend the idea of localized editing from the UI level to the model level, so that the corrected span replaces the original output in the actual context used for continuation.

G Performance of Large-Scale LLMs

In this section, we present the results obtained with large-scale LLMs. We use two open-source large-scale LLMs, such as `Llama-3.1-70B-Instruct` and `Gemma-3-27b-it`.

G.1 Task Performance

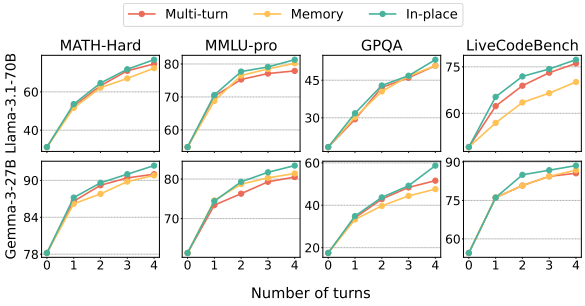

Figure G.1: Performance comparison in MATH-hard, MMLU-pro, GPQA, and LiveCodeBench. Across all datasets and large-scale LLM models, in-place feedback consistently outperforms the multi-turn and memory-based feedback approaches. Tabulated results are provided in Tables G.1 to G.4.

Figure G.1 shows the task performance, which follows a similar trend to Figure 3. While the gap between baselines and in-place feedback is smaller, this is largely because the absolute performance is already high, leaving limited room for improvement. Additionally, these LLMs have been extensively tuned via instruction-tuning to follow user instructions effectively, thereby benefiting baselines. Exploring whether dedicated post-training can further amplify the advantages of in-place feedback remains an interesting direction for future work. Appendices G.2 and G.3 show the input and generated token counts, respectively, confirming that in-place feedback maintains its token efficiency at larger scales. Overall, these results demonstrate that in-place feedback achieves the highest task performance and is the most token-efficient, even in large models.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	31.2	52.6	63.2	71.0	74.6
	Memory	31.2	51.6	62.2	67.0	72.4
	In-place	31.2	53.6	64.6	71.8	76.8
Gemma-3-27B	Multi-turn	78.2	86.6	89.2	90.4	91.0
	Memory	78.2	86.2	87.8	89.8	90.8
	In-place	78.2	87.2	89.6	91.0	92.4

Table G.1: Performance comparison on MATH-Hard.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	54.8	70.2	75.3	77.1	77.9
	Memory	54.8	68.8	76.5	78.5	80.2
	In-place	54.8	70.6	77.7	79.1	81.3
Gemma-3-27B	Multi-turn	61.3	73.4	76.3	79.3	80.5
	Memory	61.3	74.6	78.7	80.3	81.4
	In-place	61.3	74.4	79.3	81.7	83.4

Table G.2: Performance comparison MMLU-pro.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	18.3	29.4	42.1	46.0	50.8
	Memory	18.3	30.2	40.5	46.6	50.8
	In-place	18.3	31.8	42.9	46.8	53.2
Gemma-3-27B	Multi-turn	17.5	34.1	42.9	48.4	51.6
	Memory	17.5	33.3	39.7	44.4	47.6
	In-place	17.5	34.9	43.7	49.2	58.7

Table G.3: Performance comparison on GPQA.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	49.1	62.3	68.9	73.1	76.1
	Memory	49.1	56.9	63.5	66.5	70.1
	In-place	49.1	65.3	71.9	74.3	77.3
Gemma-3-27B	Multi-turn	54.5	76.1	80.8	84.4	85.6
	Memory	54.5	76.1	80.8	84.4	86.8
	In-place	54.5	76.1	85.0	86.8	88.6

Table G.4: Performance comparison on LiveCodeBench.

G.2 Number of Input Tokens

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	231.0	715.1	1373.4	2070.4	2902.1
	Memory	231.0	740.4	1059.7	1301.0	1340.6
	In-place	231.0	378.0	472.7	534.1	610.2
Gemma-3-27B	Multi-turn	177.9	1833.3	3527.8	4934.1	6250.7
	Memory	177.9	1897.3	1769.4	1958.0	2020.5
	In-place	177.9	1008.8	1425.0	1540.9	1731.2

Table G.5: Number of input tokens on Math-hard.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	165.1	747.6	1342.3	1987.7	2608.3
	Memory	165.1	776.8	954.3	1187.8	1210.4
	In-place	165.1	451.9	515.8	560.3	595.4
Gemma-3-27B	Multi-turn	149.0	894.6	1672.0	2454.1	3209.8
	Memory	149.0	924.6	1117.7	1315.8	1314.0
	In-place	149.0	473.6	553.6	642.9	649.5

Table G.6: Number of input tokens on MMLU-pro.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	224.1	998.2	1720.3	2405.4	3099.9
	Memory	224.1	1027.5	1138.1	1309.9	1307.2
	In-place	224.1	585.9	636.0	694.4	752.2
Gemma-3-27B	Multi-turn	205.0	1085.9	1903.6	2754.2	3606.0
	Memory	205.0	1115.9	1230.8	1380.2	1414.2
	In-place	205.0	555.0	692.9	756.2	783.1

Table G.7: Number of input tokens on GPQA.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	584.3	1317.0	2055.4	2835.1	3598.0
	Memory	584.3	1342.2	1439.8	1648.5	1662.7
	In-place	584.3	931.8	1007.7	1080.3	1186.6
Gemma-3-27B	Multi-turn	594.3	1251.5	1922.7	2529.5	3266.9
	Memory	594.3	1571.1	1914.6	2182.0	2254.8
	In-place	594.3	1005.6	1139.5	1180.6	1263.5

Table G.8: Number of input tokens on LiveCodeBench.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	915.3	2488.4	4109.8	5844.1	7626.1
	Memory	915.3	2454.8	2250.5	2460.7	2558.0
	In-place	915.3	2147.0	3037.9	3612.3	4322.2
Gemma-3-27B	Multi-turn	961.4	2269.8	2971.6	3663.9	4401.4
	Memory	961.4	2245.8	2133.8	2421.5	2599.9
	In-place	961.4	2067.0	2411.5	2638.4	2848.3

Table G.9: Number of input tokens on ZebraLogic.

G.3 Generated Token Length

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	279.8	380.0	482.5	522.4	593.4
	Memory	279.8	434.3	555.5	615.1	592.4
	In-place	279.8	249.2	239.0	315.2	300.5
Gemma-3-27B	Multi-turn	927.4	1134.5	1238.2	1154.2	1098.9
	Memory	927.4	969.4	1134.5	1222.9	1204.4
	In-place	927.4	807.4	731.0	879.4	756.5

Table G.10: Number of generated tokens on Math-hard.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	310.1	385.4	423.6	467.4	477.6
	Memory	310.1	432.1	515.1	575.7	623.8
	In-place	310.1	153.9	160.8	166.7	147.7
Gemma-3-27B	Multi-turn	420.3	511.5	561.6	561.3	605.3
	Memory	420.3	586.8	680.9	712.1	723.7
	In-place	420.3	291.5	297.5	327.8	324.0

Table G.11: Number of generated tokens on MMLU-pro.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	597.8	549.2	508.2	515.6	571.9
	Memory	597.8	574.9	607.8	600.5	620.3
	In-place	597.8	280.1	229.7	232.3	259.7
Gemma-3-27B	Multi-turn	738.9	641.1	643.1	666.4	679.0
	Memory	738.9	683.3	728.2	751.3	749.8
	In-place	738.9	417.1	330.4	292.9	315.6

Table G.12: Number of generated tokens on GPQA.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	467.9	488.3	513.9	553.0	517.2
	Memory	467.9	369.2	392.5	420.2	429.3
	In-place	467.9	233.2	217.7	238.8	212.5
Gemma-3-27B	Multi-turn	449.1	573.4	499.5	472.8	522.9
	Memory	449.1	552.9	704.2	835.5	895.1
	In-place	449.1	285.8	438.3	482.7	313.8

Table G.13: Number of generated tokens on LiveCodeBench.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	1429.6	1444.6	1439.1	1431.2	1445.6
	Memory	1429.6	1126.5	1200.4	1178.2	1216.5
	In-place	1429.6	1069.6	791.1	762.5	666.8
Gemma-3-27B	Multi-turn	1191.3	575.1	590.1	585.2	605.5
	Memory	1191.3	985.0	1153.2	1250.3	1313.4
	In-place	1191.3	414.2	387.7	402.7	423.0

Table G.14: Number of generated tokens on ZebraLogic.

G.4 Results on ZebraLogic

We conduct experiments on ZebraLogic with Large-scale LLMs, such as `Gemma-3-27b-it` and `Llama-3.1-70B-Instruct`. Due to cost constraints, we run five turns and one seed.

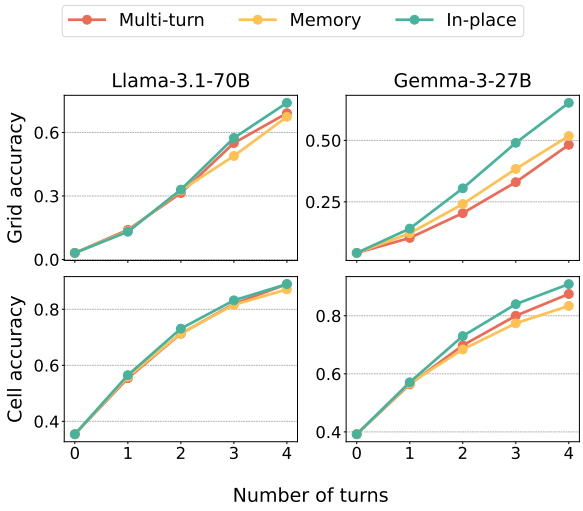

Figure G.2: Grid and cell accuracy on the ZebraLogic dataset with large-scale LLMs. Tabulated results are provided in Tables G.15 to G.18.

Task performance. As shown in Figure G.2, in-place feedback also outperforms other baselines, except for `Llama` with top-2 feedback. This gap arises from two factors: the in-place agent is less reliable than human experts at applying corrections, and due to the parallel reasoning structure of ZebraLogic, the edited span may inherit incorrect reasoning from earlier turns. This limitation could be exacerbated under top-2 feedback, in which fewer corrections lead to more errors that propagate through subsequent reasoning.

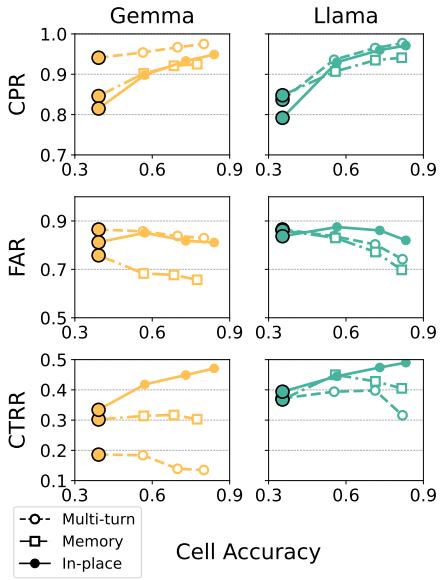

Figure G.3: CPR, FAR, and CTRR for five-turn conversations of large-scale LLMs on the ZebraLogic dataset with top-4 feedback. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

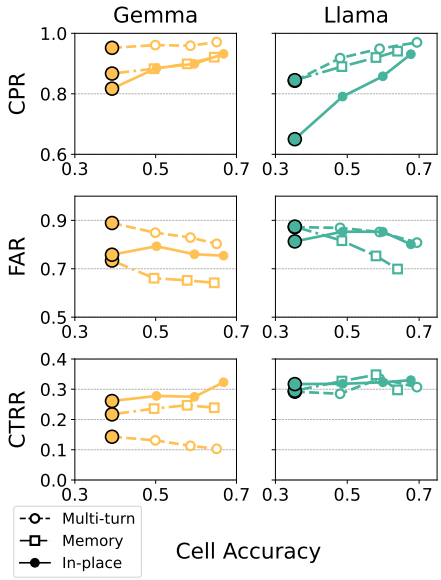

Figure G.4: CPR, FAR, and CTRR for five-turn conversations of large-scale LLMs on the ZebraLogic dataset with top-2 feedback. The points with a black border represent the second response of the LLMs (i.e., y_1), and the subsequent responses across turns are connected by lines.

Turn-level dynamics. Figures G.3 and G.4 illustrate turn-level dynamics for large-scale LLMs. The overall patterns align with Figure 5. In-place

feedback maintains the highest CTRR across all turns. In contrast, multi-turn feedback shows a rising CPR but a declining CTRR over turns, indicating a trade-off between preserving correctness and supporting further reasoning. Memory-based and in-place feedback keep CTRR relatively stable across turns, although their CPR is generally lower than that of multi-turn feedback. Unlike the smaller models, the decline in FAR across turns is less pronounced in these large-scale LLMs, likely due to improved instruction-following capability. However, FAR decline is still observable in `Llama` with top-4 feedback, suggesting that this pattern may emerge with more turns.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	0.031	0.140	0.314	0.550	0.691
	Memory	0.031	0.134	0.326	0.489	0.674
	In-place	0.031	0.131	0.330	0.574	0.740
Gemma-3-27B	Multi-turn	0.042	0.103	0.204	0.330	0.482
	Memory	0.042	0.122	0.241	0.384	0.518
	In-place	0.042	0.141	0.305	0.490	0.653

Table G.15: Grid accuracy of large-scale LLMs on ZebraLogic with top-4 feedback.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	0.031	0.096	0.195	0.337	0.492
	Memory	0.031	0.108	0.234	0.360	0.455
	In-place	0.031	0.096	0.201	0.335	0.475
Gemma-3-27B	Multi-turn	0.042	0.094	0.166	0.234	0.314
	Memory	0.042	0.099	0.180	0.262	0.344
	In-place	0.042	0.115	0.209	0.297	0.405

Table G.16: Grid accuracy of large-scale LLMs on ZebraLogic with top-2 feedback.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	0.354	0.554	0.713	0.819	0.890
	Memory	0.354	0.559	0.714	0.816	0.871
	In-place	0.354	0.565	0.731	0.832	0.890
Gemma-3-27B	Multi-turn	0.392	0.564	0.698	0.800	0.875
	Memory	0.392	0.567	0.684	0.774	0.834
	In-place	0.392	0.571	0.730	0.840	0.909

Table G.17: Cell accuracy of large-scale LLMs on ZebraLogic. Top-4 feedback is used.

Model	Method	Number of turns				
		0	1	2	3	4
Llama-3.1-70B	Multi-turn	0.354	0.480	0.590	0.694	0.756
	Memory	0.354	0.485	0.580	0.640	0.696
	In-place	0.354	0.487	0.599	0.677	0.730
Gemma-3-27B	Multi-turn	0.392	0.499	0.586	0.651	0.711
	Memory	0.392	0.496	0.578	0.645	0.699
	In-place	0.392	0.502	0.596	0.668	0.747

Table G.18: Cell accuracy of large-scale LLMs on ZebraLogic with top-2 feedback.

H Tabulated Results

This section provides the numerical values corresponding to the main figures presented in the paper.

H.1 Task Performance

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Qwen	Multi-turn	55.0	68.4	73.8	76.6	79.2	81.6	83.8	85.0	85.8	86.2
	Memory	55.0	62.6	66.2	69.2	71.2	72.8	74.2	75.0	76.4	77.4
	In-place	55.0	71.4	76.6	81.2	85.2	88.8	90.4	91.8	92.4	92.8
Gemma	Multi-turn	49.2	60.6	68.4	73.0	76.0	77.6	79.2	80.4	82.0	82.8
	Memory	49.2	60.4	65.0	68.0	70.6	72.6	74.6	76.4	77.2	78.0
	In-place	49.2	62.6	70.8	74.8	77.8	81.4	82.8	85.0	86.6	87.6
Llama	Multi-turn	14.8	34.0	45.8	56.4	59.8	64.4	67.4	71.2	73.4	75.6
	Memory	14.8	29.4	40.0	47.0	52.8	57.2	62.2	65.6	67.6	69.8
	In-place	14.8	36.6	48.0	59.8	66.8	74.4	76.8	80.6	82.8	84.0

Table H.1: Performance comparison on Math-hard.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Qwen	Multi-turn	56.2	68.0	72.6	75.1	77.3	78.1	79.1	79.3	80.1	80.5
	Memory	56.2	67.1	70.8	73.4	75.7	77.7	78.5	79.7	80.7	80.9
	In-place	56.2	69.8	78.5	80.1	81.9	83.0	83.8	84.6	85.0	85.4
Gemma	Multi-turn	41.4	54.0	60.0	65.9	68.4	69.8	71.2	72.0	72.8	73.4
	Memory	41.4	52.9	57.2	63.5	66.3	68.0	69.6	70.4	71.6	72.2
	In-place	41.4	59.4	65.5	69.6	73.6	76.1	77.7	79.5	80.7	81.3
Llama	Multi-turn	42.2	55.8	61.5	65.7	69.0	71.2	72.0	73.4	74.8	75.1
	Memory	42.2	57.0	63.9	68.6	71.2	74.0	75.9	77.3	78.3	78.7
	In-place	42.2	59.4	68.2	74.2	76.5	79.7	81.3	82.8	84.2	84.8

Table H.2: Performance comparison on MMLU-pro.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Qwen	Multi-turn	10.3	17.5	27.8	38.1	40.5	42.9	43.7	43.7	43.7	46.0
	Memory	10.3	19.0	27.0	29.4	34.9	36.5	37.3	38.1	38.1	38.9
	In-place	10.3	25.4	36.5	48.4	51.6	57.9	63.5	64.3	66.7	69.0
Gemma	Multi-turn	5.6	9.5	11.9	14.3	15.1	19.0	22.2	23.0	24.6	27.0
	Memory	5.6	8.7	10.3	13.5	15.9	17.5	18.3	22.2	22.2	23.8
	In-place	5.6	11.9	27.8	33.3	38.9	42.9	46.0	46.0	47.6	53.2
Llama	Multi-turn	4.0	8.7	19.0	24.6	27.0	31.0	34.1	38.1	42.1	43.7
	Memory	4.0	8.7	15.1	20.6	23.0	27.8	31.7	34.1	37.3	40.5
	In-place	4.0	15.1	26.2	35.7	43.7	47.6	53.2	55.6	61.1	61.9

Table H.3: Performance comparison on GPQA.

H.2 Number of Input Tokens

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	231.0	800.9	1572.4	2453.3	3513.4	4433.9	5513.2	6621.6	7854.8	8949.1
	Memory	231.0	826.2	1073.4	1319.3	1361.2	1414.1	1413.2	1468.8	1515.9	1566.9
	In-place	231.0	373.3	456.5	533.1	608.1	666.8	733.2	790.0	825.0	872.1
Gemma	Multi-turn	165.9	1822.6	3247.8	4640.9	5850.4	7188.1	8483.4	9833.5	11076.5	12444.9
	Memory	165.9	1898.6	1537.5	1712.3	1717.8	1737.9	1712.2	1744.5	1739.2	1720.0
	In-place	165.9	770.8	1042.0	1316.6	1334.0	1355.4	1439.0	1635.8	1683.7	1727.9
Qwen	Multi-turn	159.6	1288.6	2538.6	3838.7	5235.0	6806.2	8368.0	9977.5	11671.0	13434.7
	Memory	159.6	1364.8	858.4	1134.3	1100.5	1144.9	1153.2	1160.6	1095.7	1125.5
	In-place	159.6	734.7	950.6	1088.9	1135.6	1250.6	1302.8	1392.1	1439.3	1464.7

Table H.4: Number of input tokens on Math-hard.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	165.1	800.4	1473.2	2164.4	2871.0	3594.4	4274.1	4962.2	5578.6	6248.8
	Memory	165.1	829.6	973.4	1157.5	1209.8	1224.9	1256.0	1237.0	1249.4	1279.0
	In-place	165.1	422.1	487.9	534.0	592.6	625.2	665.0	638.6	677.9	670.8
Gemma	Multi-turn	149.0	975.2	1770.5	2545.2	3392.5	4187.6	4961.9	5791.9	6656.0	7565.4
	Memory	149.0	1005.2	1078.5	1191.2	1210.4	1217.3	1256.2	1272.8	1267.5	1285.5
	In-place	149.0	410.0	579.0	658.5	750.2	830.5	841.1	802.7	863.0	863.6
Qwen	Multi-turn	144.8	918.2	1809.5	2724.6	3698.0	4786.0	5804.9	6891.6	7984.2	8977.5
	Memory	144.8	947.4	1125.8	1282.0	1318.7	1333.7	1367.8	1345.4	1415.4	1449.1
	In-place	144.8	519.4	625.5	764.1	794.8	833.9	807.9	861.3	886.9	882.1

Table H.5: Number of input tokens on MMLU-pro.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	224.1	1054.1	1859.6	2744.6	3635.5	4503.7	5393.5	6302.9	7250.4	8134.1
	Memory	224.1	1083.4	1181.7	1394.7	1389.8	1360.0	1377.9	1354.2	1319.9	1340.8
	In-place	224.1	522.0	611.1	689.5	725.9	740.0	770.9	806.7	890.8	876.4
Gemma	Multi-turn	205.0	1197.2	2138.4	3073.1	3996.3	4913.9	5850.2	6865.6	7829.9	8823.9
	Memory	205.0	1227.2	1322.4	1482.0	1466.4	1508.8	1522.3	1498.5	1524.4	1503.4
	In-place	205.0	480.3	666.0	680.4	755.3	848.7	932.5	848.2	1004.4	980.0
Qwen	Multi-turn	203.6	1079.9	2013.6	2982.9	4127.7	5304.3	6395.3	7570.8	8763.9	9987.4
	Memory	203.6	1109.2	1258.6	1484.6	1521.4	1541.4	1544.0	1519.1	1511.7	1525.8
	In-place	203.6	553.7	678.3	771.6	759.5	766.1	847.4	848.5	876.1	966.4

Table H.6: Number of input tokens on GPQA.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	932.3	2748.8	4089.0	5760.6	7508.6	9251.7	11123.1	12914.4	14775.8	16653.6
	Memory	932.3	2700.9	2275.6	2312.4	2370.3	2402.9	2433.7	2439.4	2434.0	2386.2
	In-place	932.3	2166.4	2709.9	3171.8	3641.0	4069.7	4551.9	5079.0	5519.2	5878.4
Gemma	Multi-turn	1004.4	2249.6	3464.7	4709.9	5963.8	7295.7	8601.6	9843.2	11092.1	12269.7
	Memory	1004.4	2171.6	2341.8	2414.9	2479.8	2470.6	2490.3	2520.4	2534.1	2525.8
	In-place	1004.4	1969.5	2286.8	2610.3	2935.5	3262.7	3544.8	3786.9	3983.6	4258.2
Qwen	Multi-turn	897.8	2144.7	3342.6	4579.8	5844.5	7108.0	8416.9	9860.3	11128.0	12343.3
	Memory	897.8	2128.5	2152.0	2196.5	2212.2	2190.5	2242.1	2234.0	2217.8	2277.1
	In-place	897.8	1878.6	2100.1	2346.6	2635.3	2941.3	3184.2	3381.4	3511.4	3493.4

Table H.7: Number of input tokens on ZebraLogic.

H.3 Number of Generated Tokens

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	395.2	504.4	587.3	661.0	730.8	734.3	792.9	809.8	852.6	923.6
	Memory	395.2	479.1	588.1	647.8	696.7	705.3	743.1	790.1	844.8	861.5
	In-place	395.2	294.1	307.8	331.9	328.6	351.5	373.8	388.6	364.8	432.4
Gemma	Multi-turn	1138.3	1152.2	1098.2	1074.0	1070.4	1087.0	1063.4	1091.2	1081.9	1119.2
	Memory	1138.3	907.0	961.6	986.1	987.7	984.3	999.7	1041.3	1013.6	992.2
	In-place	1138.3	1042.9	838.6	746.1	761.0	758.3	681.3	617.9	554.4	583.3
Qwen	Multi-turn	810.9	925.4	1072.1	1186.2	1276.7	1349.3	1360.0	1408.0	1469.4	1525.9
	Memory	810.9	850.7	945.5	987.7	1011.2	1020.3	1033.4	1031.4	1050.0	1071.5
	In-place	810.9	403.1	320.2	277.7	283.0	232.6	224.2	172.0	166.8	185.7

Table H.8: Number of generated tokens on Math-hard.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	381.4	432.4	477.4	480.0	499.4	537.3	537.0	514.7	518.2	546.3
	Memory	381.4	429.7	515.0	574.6	603.1	627.7	617.2	650.2	662.3	642.3
	In-place	381.4	236.2	236.7	206.9	214.0	212.8	188.0	249.9	223.5	253.1
Gemma	Multi-turn	551.7	567.9	562.3	585.7	603.2	635.0	641.1	662.3	683.4	683.2
	Memory	551.7	564.8	600.9	593.8	614.2	655.1	664.4	667.3	688.2	696.4
	In-place	551.7	505.6	475.5	468.4	398.9	414.5	395.9	455.7	399.0	384.0
Qwen	Multi-turn	456.1	614.2	689.3	757.0	813.6	851.4	860.8	907.1	919.7	943.1
	Memory	456.1	597.1	678.3	721.4	736.5	780.6	766.5	794.5	847.6	833.7
	In-place	456.1	280.4	305.4	265.8	269.9	289.9	336.0	298.2	320.2	308.2

Table H.9: Number of generated tokens on MMLU-pro.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	665.7	644.4	659.2	683.2	721.5	699.2	697.8	689.6	697.8	690.8
	Memory	665.7	614.8	671.1	678.5	667.1	673.4	660.3	646.5	651.6	627.4
	In-place	665.7	421.3	351.0	283.8	312.3	297.0	292.7	306.9	241.3	266.8
Gemma	Multi-turn	822.6	811.8	756.1	731.8	773.5	736.9	796.8	809.3	806.9	820.0
	Memory	822.6	796.5	797.6	774.2	815.5	837.2	816.6	835.6	823.3	815.7
	In-place	822.6	625.9	514.9	533.5	516.3	472.7	376.3	462.9	315.5	386.8
Qwen	Multi-turn	715.5	751.6	832.8	860.5	946.8	964.7	1040.5	1035.5	1067.5	1068.6
	Memory	715.5	709.6	787.0	824.5	847.2	853.0	835.1	815.3	837.1	850.9
	In-place	715.5	399.5	332.2	270.9	272.5	275.7	245.1	253.2	247.8	192.1

Table H.10: Number of generated tokens on GPQA.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Llama	Multi-turn	1685.8	1237.7	1502.3	1597.5	1535.5	1558.8	1558.4	1623.8	1572.2	1600.6
	Memory	1685.8	1181.6	1118.8	1164.6	1182.5	1176.5	1175.7	1169.2	1133.1	1144.5
	In-place	1685.8	746.2	566.9	523.3	501.3	528.7	529.6	563.8	582.7	662.8
Gemma	Multi-turn	1123.1	1073.6	1090.5	1096.2	1104.9	1118.8	1129.5	1140.4	1147.1	1143.0
	Memory	1123.1	1207.0	1194.1	1263.0	1251.1	1271.5	1299.5	1313.1	1304.8	1318.6
	In-place	1123.1	454.8	427.6	425.8	427.7	436.6	454.6	479.6	477.8	510.7
Qwen	Multi-turn	1135.0	1081.0	1085.3	1089.1	1104.9	1117.3	1130.4	1152.4	1158.7	1169.2
	Memory	1135.0	1074.9	1021.9	1019.1	1006.9	1024.0	1029.1	996.6	1039.3	1071.6
	In-place	1135.0	333.1	322.4	333.0	358.4	381.3	403.5	415.0	416.9	405.1

Table H.11: Number of generated tokens on ZebraLogic.

H.4 Results on ZebraLogic

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Gemma	Multi-turn	0.000	0.028	0.078	0.148	0.271	0.421	0.556	0.650	0.728	0.771
	Memory	0.000	0.007	0.028	0.052	0.078	0.105	0.135	0.167	0.199	0.234
	In-place	0.000	0.031	0.115	0.265	0.441	0.582	0.712	0.816	0.868	0.906
Qwen	Multi-turn	0.003	0.022	0.071	0.150	0.249	0.356	0.468	0.566	0.645	0.690
	Memory	0.004	0.025	0.073	0.166	0.251	0.348	0.434	0.520	0.600	0.665
	In-place	0.004	0.036	0.144	0.341	0.534	0.672	0.779	0.850	0.895	0.922
Llama	Multi-turn	0.006	0.017	0.056	0.126	0.225	0.358	0.470	0.551	0.612	0.652
	Memory	0.006	0.023	0.076	0.160	0.264	0.369	0.457	0.543	0.625	0.682
	In-place	0.006	0.034	0.105	0.223	0.359	0.478	0.590	0.673	0.738	0.790

Table H.12: Grid accuracy of LLMs on ZebraLogic with top-4 feedback.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Gemma	Multi-turn	0.000	0.013	0.035	0.070	0.109	0.156	0.208	0.266	0.325	0.390
	Memory	0.000	0.005	0.018	0.033	0.044	0.058	0.078	0.089	0.101	0.120
	In-place	0.000	0.013	0.057	0.103	0.167	0.252	0.334	0.421	0.494	0.582
Qwen	Multi-turn	0.004	0.013	0.029	0.059	0.093	0.136	0.188	0.229	0.276	0.322
	Memory	0.004	0.015	0.032	0.071	0.119	0.157	0.217	0.278	0.342	0.395
	In-place	0.004	0.020	0.048	0.114	0.194	0.286	0.382	0.471	0.552	0.624
Llama	Multi-turn	0.006	0.018	0.040	0.073	0.125	0.183	0.250	0.324	0.385	0.433
	Memory	0.005	0.017	0.044	0.089	0.143	0.193	0.248	0.303	0.339	0.385
	In-place	0.006	0.021	0.060	0.110	0.174	0.258	0.328	0.391	0.448	0.502

Table H.13: Grid accuracy of LLMs on ZebraLogic with top-2 feedback.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Gemma	Multi-turn	0.272	0.435	0.567	0.681	0.779	0.848	0.896	0.927	0.947	0.959
	Memory	0.272	0.350	0.411	0.464	0.505	0.528	0.548	0.571	0.586	0.603
	In-place	0.272	0.432	0.573	0.698	0.794	0.859	0.914	0.944	0.963	0.972
Qwen	Multi-turn	0.263	0.417	0.543	0.652	0.736	0.806	0.859	0.891	0.921	0.935
	Memory	0.264	0.406	0.518	0.618	0.690	0.750	0.794	0.835	0.866	0.891
	In-place	0.264	0.443	0.607	0.738	0.830	0.891	0.931	0.956	0.968	0.977
Llama	Multi-turn	0.191	0.185	0.366	0.460	0.546	0.612	0.671	0.663	0.701	0.718
	Memory	0.191	0.339	0.487	0.599	0.693	0.758	0.807	0.841	0.871	0.889
	In-place	0.191	0.367	0.480	0.591	0.679	0.751	0.798	0.833	0.847	0.883

Table H.14: Cell accuracy of LLMs on ZebraLogic with top-4 feedback.

	Method	Number of turns									
		0	1	2	3	4	5	6	7	8	9
Gemma	Multi-turn	0.272	0.367	0.441	0.511	0.575	0.629	0.679	0.723	0.761	0.793
	Memory	0.272	0.311	0.347	0.376	0.401	0.428	0.445	0.461	0.482	0.501
	In-place	0.272	0.359	0.434	0.510	0.580	0.642	0.702	0.754	0.793	0.832
Qwen	Multi-turn	0.264	0.350	0.421	0.482	0.538	0.585	0.629	0.666	0.698	0.730
	Memory	0.264	0.339	0.410	0.471	0.529	0.581	0.628	0.669	0.703	0.735
	In-place	0.264	0.361	0.452	0.538	0.614	0.682	0.742	0.789	0.828	0.858
Llama	Multi-turn	0.191	0.289	0.331	0.404	0.452	0.514	0.543	0.595	0.608	0.649
	Memory	0.195	0.283	0.373	0.458	0.517	0.573	0.609	0.637	0.668	0.705
	In-place	0.191	0.311	0.376	0.451	0.506	0.566	0.605	0.645	0.674	0.705

Table H.15: Cell accuracy of LLMs on ZebraLogic with top-2 feedback.

I Qualitative Examples

I.1 In-Place Feedback Example

Figures I.9 to I.12 are the qualitative examples of in-place feedback on the three benchmarks.

I.2 Multi-Turn Feedback Failure Example

We observe failure cases of multi-turn feedback, and present the instances in Figures I.13 to I.15.

J User Study Interface

Figures J.1 to J.7 are examples of our user study interface.

Prompt format used for memory-based feedback

USER PROMPT
Question: {QUESTION}
Previous reasoning response: {PREVIOUS RESPONSE}
Accumulated Feedback: {ACCUMULATED FEEDBACK}
Please revise your previous reasoning response based on the accumulated feedback.

Figure I.1: Prompt format used for memory-based feedback in MATH-hard, MMLU-pro free-form, GPQA free-form, and LiveCodeBench. The system prompt is identical to that in multi-turn feedback.

LLM-as-a-judge prompt for MATH-hard

SYSTEM PROMPT
You are a mathematical equivalence judge.
Given a question, a correct answer, and a model's prediction, determine if they are mathematically equivalent even if they have different formatting or representation.
Respond with ONLY 'YES' if they are equivalent, or 'NO' if they are not.
If the model's prediction is correct, but the formatting is wrong, please respond with 'YES'.
DO NOT RESPOND WITH ANYTHING ELSE.

USER PROMPT
Question: {QUESTION}
Correct answer: {CORRECT_ANSWER}
Model prediction: {PREDICTION}
Are these equivalent? Answer YES or NO.

(a) Prompt format used in MATH-hard.

LLM-as-a-judge prompt for MMLU-pro free-form and GPQA free-form

SYSTEM PROMPT
You are a strict and impartial judge for evaluating model predictions.
Given a question, a correct answer, and a model's prediction, decide whether the prediction is equivalent to the correct answer.
Respond with ONLY 'YES' or 'NO'. Never output anything else.

Judgment Criteria
1. The prediction must fully answer the question and cannot be vague.
2. Ignore differences in formatting, punctuation, capitalization, or spacing.
3. For numeric answers:
- Compute the relative error = |prediction − ground truth| / mean(prediction, ground truth).
- The prediction is correct if the relative error $< 1\%$.
- If the ground truth is a single numeric value but the prediction is a range, treat it as incorrect (even if the range contains the ground truth).
- If the ground truth is a single value but the prediction is a range, output 'NO'.
4. If the prediction is correct in substance but expressed in a different format, respond 'YES'.

Final Rule: Output must be exactly 'YES' or 'NO'. No explanations, no extra words.

USER PROMPT
Question: {QUESTION}
Correct answer: {CORRECT_ANSWER}
Model prediction: {PREDICTION}
Are these equivalent? Answer YES or NO.

(b) Prompt format used in MMLU-pro free-form and GPQA free-form.

Figure I.2: Prompt format used for LLM-as-a-judge in MATH-hard, MMLU-pro free-form, and GPQA free-form.

Prompt format used for feedback agent in MATH-hard, MMLU-pro free-form, and GPQA free-form

SYSTEM PROMPT
You are a rigorous verifier of LLM answers. Your task is to identify the first and most critical reasoning flaw that causes the error, given a question, the model's full answer, and the correct answer.
You MUST NOT reveal or speculate about the correct final answer.
- Note that the options in previous questions might have been switched in each different attempt.
Your output must follow the structure below:
Explain in (maximum 2–3) precise sentences why that step is incorrect. Be specific in your critique.
Give short and specific actionable advice to fix it.
Constraints:
You may only give feedback on one reasoning error at a time — the earliest critical one.
If the model's answer is correct in logic but wrong in formatting, your feedback should address the formatting issue.
You DO NOT leak and mention the correct final answer.
DO NOT SOLVE THE PROBLEM, JUST PROVIDE FEEDBACK WITHOUT MENTIONING THE CORRECT ANSWER
Your output strictly starting with "Your output is wrong because" and then followed by the explanation.
Do not output anything else.

USER PROMPT
Your goal is to identify the **earliest and most critical logical flaw** in the model's reasoning
DO NOT provide or hint at the correct final answer.
- Note that the options in previous questions might have been switched in each different attempt.
Problem
{PROBLEM}
Most Recent Model Answer
{MODEL ANSWER}
Correct Final Answer
{GROUND TRUTH}
Based on this, provide feedback on the single most important error in the model's answer.
Do not leak and mention the correct final answer and do not add any extra commentary.

(a) Prompt format used for feedback agent in MATH-hard, MMLU-pro free-form, and GPQA free-form.

Prompt format used for feedback agent in LiveCodeBench

SYSTEM PROMPT
You are a rigorous verifier of coding solutions. Your task is to identify the first and most critical logical flaw in the code that causes test failures, given a coding problem, the model's submitted code.
You MUST NOT provide the complete corrected code.
Your output must follow this structure:
1. Start with "Your output is wrong because"
2. Explain what is wrong
3. End with how to fix it.
Example:
"Your output is wrong because the loop breaks after finding the first valid partition, but the problem requires finding the minimum among all possible partitions.
Iterate through all possible split points and track the minimum maximum sum using min()."
Constraints:
You may only give feedback on one coding error at a time — the earliest critical one.
Be specific about which part of the code is problematic (mention line numbers or code patterns if helpful).
Do not mention the complete implementation.
DO NOT provide the full corrected code, and do not hard-code the example input/output in the code.
Do not output anything else.

USER PROMPT
Your goal is to identify the **earliest and most critical logical flaw** in the model's reasoning
DO NOT provide the complete solution.
- Note that the options in previous questions might have been switched in each different attempt.
Problem
{PROBLEM}
Most Recent Model Answer
{MODEL ANSWER}

(b) Prompt format used in LiveCodeBench.

Figure I.3: Prompt format used for the feedback agent in empirical experiments.

Prompt format used for in-place feedback agent in MATH-hard, MMLU-pro free-form, and GPQA free-form

SYSTEM PROMPT
You are a helpful assistant who INTERVENES in a math solution based on user feedback.
Your job is to produce a JSON object for a single **in-place replace** operation:
- Identify the **shortest unique substring (SUS)** from the original solution that must be edited to apply the feedback.
- Produce the revised text for that exact span.
- **Do NOT** change anything before the flaw, and **do NOT** continue solving the problem beyond where the feedback applies.
- **Preserve** all whitespace, punctuation, LaTeX, and casing exactly as in the original solution for the target substring.
- The target must be a **contiguous** substring that occurs **exactly once** in the original solution. If not unique, minimally extend the span (e.g., include adjacent tokens or punctuation) until it becomes unique.
- Return **ONLY** valid JSON with UTF-8 and proper escaping (no trailing commas, no extra commentary).

Return JSON with this schema (single edit only):

```
{
    "target_sentence": "< exact shortest unique substring copied from the original>",
    "edit_sentence": "<the revised substring after applying the feedback>"
}
```

Constraints:
- Output must be a single-line or multi-line JSON object; do not include any extra text.
- Do not normalize quotes/hyphens/spaces; copy exactly from the original for target_sentence.
- Do not introduce additional edits beyond the specified span.
- Do not provide a reasoning process beyond the feedback.

USER PROMPT
<The Start of Answer>
{ANSWER}
<The End of Answer>
<The Start of Original Solution>
{ORIGINAL_SOLUTION}
<The End of Original Solution>
<The Start of User Feedback>
{USER_FEEDBACK}
<The End of User Feedback>
<The Start of Instructions>
Write the JSON according to the following:
- Apply ONLY the given feedback to the original solution.
- Identify the **shortest unique substring** in the original that must change to satisfy the feedback; this must appear **exactly once**.
- If the obvious sentence occurs multiple times, **minimally extend** the span (e.g., prepend/append one or two nearby tokens or punctuation) until uniqueness holds.
- Put the original substring in "target_sentence" (copied **verbatim** from the original, including whitespace/newlines).
- Put the corrected version in "edit_sentence".
- If the feedback is sentence-like, keep it within **three sentences** in the edited span.
- STRICTLY FOLLOW:
- **Do not solve the problem** beyond where the feedback applies.
- Stop right after applying the feedback.
- Return **ONLY** valid JSON with keys "target_sentence" and "edit_sentence".

Figure I.4: Prompt format used for in-place feedback agent in MATH-hard, MMLU-pro free-form, and GPQA free-form.

Prompt format used for in-place feedback agent in LiveCodeBench

SYSTEM PROMPT
You are a rigorous verifier of code solutions. Your job is to produce a JSON object for a single **in-place replace** operation:
- Identify the **shortest unique substring (SUS)** from the original solution that must be edited to apply the feedback.
- Produce the revised text for that exact span.
- The edit_sentence must contain the whole feedback information, do not miss any information.
- Ensure that the edit_sentence matches the style and tone of the original solution, making it plausible to fit as a replacement.
- **Do NOT**change anything before the flaw, and **do NOT** continue solving the problem beyond where the feedback applies.
- **Preserve** all whitespace, indentation, and casing exactly as in the original solution for the target substring.
- The target must be a **contiguous** substring that occurs **exactly once** in the original solution.

Important for code continuation:
- The edit_sentence should end in a way that naturally invites continuation. Avoid ending with a complete function or return statement.
- If the original flaw occurs mid-code, the edit_sentence should correct the error and set up the next logical step without completing it.
- Prefer ending with incomplete structures (e.g., opening a loop, starting a conditional, beginning a function body, or a partial expression).
- Do NOT include any content after the corrected span; the model will generate the continuation.

Return JSON with this schema (single edit only):

```
{
    "target_sentence": "< exact shortest unique substring copied from the original>",
    "edit_sentence": "<the revised substring after applying the feedback>"
}
```

Constraints:
- Do not normalize quotes/hyphens/spaces; copy exactly from the original for target_sentence.
- The edit_sentence must contain the whole feedback information, do not miss any information.
- Ensure that the edit_sentence matches the style and tone of the original solution, making it plausible to fit as a replacement.
- Do not provide reasoning process beyond the feedback.
- The edit_sentence should be written as much as possible in code, not in comments. Only include comments when absolutely necessary for guidance.

USER PROMPT
<The Start of Answer>
{ANSWER}
<The End of Answer>
<The Start of Original Solution>
{ORIGINAL_SOLUTION}
<The End of Original Solution>
<The Start of User Feedback>
{USER_FEEDBACK}
<The End of User Feedback>
<The Start of Instructions>
Write the JSON according to the following:
- Apply ONLY the given feedback to the original solution.
- Identify the **shortest unique substring** in the original that must change to satisfy the feedback; this must appear **exactly once**.
- If the obvious sentence occurs multiple times, **minimally extend** the span (e.g., prepend/append one or two nearby tokens or punctuation) until uniqueness holds.
- Put the original substring in "target_sentence" (copied **verbatim** from the original, including whitespace/newlines).
- Put the corrected version in "edit_sentence".
- If the feedback is sentence-like, keep it within **three sentences** in the edited span.
- STRICTLY FOLLOW:
- **Do not solve the problem** beyond where the feedback applies.
- Stop right after applying the feedback.
- Return **ONLY** valid JSON with keys "target_sentence" and "edit_sentence".

Figure I.5: Prompt format used for in-place feedback agent in LiveCodeBench.

Prompt format used for ZebraLogic

SYSTEM PROMPT
You are a helpful assistant that solves zebra puzzles.
Given a puzzle and a json template, you need to solve the puzzle and fill in the json template.
You need to fill in the json template with the correct attributes.

USER PROMPT
Example Puzzle

There are 3 houses, numbered 1 to 3 from left to right, as seen from across the street. Each house is occupied by a different person.
Each house has a unique attribute for each of the following characteristics:
- Each person has a unique name: 'Peter', 'Eric', 'Arnold'.
- Each person has a unique favorite drink: 'tea', 'water', 'milk'

Clues for the Example Puzzle

1. Peter is in the second house.
2. Arnold is directly left of the one who only drinks water.
3. The one who only drinks water is directly left of the person who likes milk.

Answer to the Example Puzzle

```
{{
    "solution": {{
        "House 1": {{
            "Name": "Arnold",
            "Drink": "tea"
        }},
        "House 2": {{
            "Name": "Peter",
            "Drink": "water"
        }},
        "House 3": {{
            "Name": "Eric",
            "Drink": "milk"
        }}
    }}
}}
```

Puzzle to Solve

{PUZZLE}

Instruction

Now please solve the above puzzle. Present your reasoning and solution in the following json format:
{JSON TEMPLATE}

Figure I.6: Input template used for ZebraLogic.

Feedback template for `Gemma` and `Qwen` in ZebraLogic

Your answer is incorrect. Please revise your solution based on the following feedback.
- '{CATEGORY}' of the '{HOUSE}' is '{GROUND TRUTH}', not '{PREDICTION}'.
- · · ·

(a) Feedback template for `Gemma` and `Qwen`.

Feedback template for `Llama` in ZebraLogic

Please revise your step-by-step reasoning based on the following feedback, and then provide a solution in the following json format.
Do not just provide the final solution, but also provide the reasoning process.
- '{CATEGORY}' of the '{HOUSE}' is '{GROUND TRUTH}', not '{PREDICTION}'.
- · · ·

(b) Feedback template for `Llama`.

Figure I.7: Rule-based feedback template in ZebraLogic.

Prompt format used for in-place feedback agent in ZebraLogic

SYSTEM PROMPT
You are an in-place patcher. Given a reasoning step and feedback items, decide if the step conflicts with the feedback. If so, produce the minimally edited step that preserves all non-conflicting content and enforces the feedback exactly.

LOCKED CLUE SPANS
- Any substring beginning with 'Clue' and ending at the first period is read-only. Do not alter or delete it.
- If a step contains locked span(s), edit only after the last locked span.
- If formatted as "Clue N: ... . - editable text", edit only after the first ' - '. If editable text is deleted, remove the dangling ' - '.

REFERENCE ATTRIBUTES
{CATEGORY}
- Use this only as a reference to detect conflicts and apply equivalence normalization. Do not generate new information beyond feedback.

EDIT RULES
- Feedback is always given in positive-only form, e.g., 'attr' of the 'house i' is 'A'.
- Always overwrite the attribute with the provided value A.
- Modify only attributes mentioned in the feedback.
- Do not add new reasoning, deductions, constraints, or explanations.
- Do not insert additional sentences beyond the minimal correction.
- Do not add new information, negations, or style fixes.
- Equivalence normalization:
* 'name' of house i is X ↔ X is in House
* 'musicgenre' of house i is Y, The Y house is House i
* Apply the same for other attributes.
- Rewrite conflicts to the positive corrected assignment. Never use negations.

CONFLICT CRITERIA
A statement conflicts only if it:
- Mentions the same (house, attribute) with a contradictory value, or
- Assigns a feedback value to the wrong house by equivalence.

DEPENDENCY PRUNING
- If a sentence depends on a premise that is corrected or negated by the feedback and cannot be minimally rewritten, delete that sentence only.
- Do not delete the entire reasoning step.
- Never alter or delete locked clue text.

OUTPUT
- If no conflict: "conflict":"no"
- If conflict: "conflict":"yes","replacement":"..."
where replacement is the fully patched step, including unchanged locked clues.

RULES
- Make the smallest edits necessary.
- Preserve capitalization, spacing, and order.
- Never output empty replacement if a locked clue exists.
- Absolutely never add new reasoning, deductions, or commentary not present in the original.

USER PROMPT
Reasoning step
{REASONING_STEP}
Feedback
{FEEDBACK}

Figure I.8: Prompt format used for in-place feedback agent in ZebraLogic.

In-place feedback example - MATH-hard

Problem
Express the infinite series $\frac{3}{206} + \frac{9}{2\cdot 103^2} + \frac{27}{2\cdot 103^3} + \cdots$ as a terminating decimal.

Model response
Let $S = \frac{3}{206} + \frac{9}{2\cdot 103^2} + \frac{27}{2\cdot 103^3} + \cdots$. We can rewrite this as

$$S = \sum_{n=1}^{\infty} \frac{3^n}{2\cdot 103^{n+1}} = \frac{1}{2}\sum_{n=1}^{\infty} \frac{3^n}{103^{n+1}} = \frac{1}{2}\sum_{n=1}^{\infty} \frac{3^n}{103\cdot 103^n} = \frac{1}{2\cdot 103}\sum_{n=1}^{\infty}\left(\frac{3}{103}\right)^n$$

The sum $\sum_{n=1}^{\infty}\left(\frac{3}{103}\right)^n$ is a geometric series with first term $\frac{3}{103}$ and common ratio $\frac{3}{103}$. Since $\left|\frac{3}{103}\right| < 1$, the series converges to

$$\sum_{n=1}^{\infty}\left(\frac{3}{103}\right)^n = \frac{\frac{3}{103}}{1-\frac{3}{103}} = \frac{\frac{3}{103}}{\frac{103-3}{103}} = \frac{\frac{3}{103}}{\frac{100}{103}} = \frac{3}{100}$$

Therefore, $S = \frac{1}{2\cdot 103}\cdot\frac{3}{100} = \frac{3}{20600}$ Now we need to express $\frac{3}{20600}$ as a terminating decimal. We can divide 3 by 20600: $\frac{3}{20600} = 0.00014645\ldots$ We can write $\frac{3}{20600} = \frac{3}{206\cdot 100} = \frac{3}{206}\cdot\frac{1}{100}$. We have $\frac{3}{206} \approx 0.01464539$. Then $\frac{3}{20600} = \frac{3}{206}\cdot\frac{1}{100} \approx 0.01464539\cdot\frac{1}{100} = 0.0001464539$. We can also write $\frac{3}{20600} = \frac{3}{206\cdot 100} = \frac{3}{206}\cdot\frac{1}{100}$. We can divide 3 by 206 to get $0.01464539\ldots$. Then we multiply by $\frac{1}{100}$ to get $0.0001464539\ldots$. We can write $\frac{3}{20600} = 0.0001464539\ldots$ The decimal representation is $0.0001464539\ldots$. We want to express this as a terminating decimal. We have $S = \frac{3}{20600}$. Dividing 3 by 20600, we get $3 \div 20600 = 0.0001464539\ldots$ Since we want a terminating decimal, we need to find a common denominator. We have $20600 = 2^3\cdot 5^2\cdot 103$. So $\frac{3}{20600} = \frac{3}{2^3\cdot 5^2\cdot 103}$. We can write $\frac{3}{20600} = 0.0001464539\ldots$ The decimal representation is $0.0001464539\ldots$. Final Answer: The final answer is $\boxed{0.000146}$

---

1) In-place edit
Let $S = \frac{3}{206} + \frac{9}{2\cdot 103^2} + \frac{27}{2\cdot 103^3} + \cdots$. We can rewrite this as $S = \sum_{n=1}^{\infty}\frac{3^n}{2\cdot 103^n} = \frac{1}{2}\sum_{n=1}^{\infty}\left(\frac{3}{103}\right)^n$

---

2) Continuation generation
This is a geometric series with first term $a = \frac{3}{103}$ and common ratio $r = \frac{3}{103}$. Since $|r| = \frac{3}{103} < 1$, the series converges and its sum is given by

$$\sum_{n=1}^{\infty}\left(\frac{3}{103}\right)^n = \frac{\frac{3}{103}}{1-\frac{3}{103}} = \frac{\frac{3}{103}}{\frac{103-3}{103}} = \frac{\frac{3}{103}}{\frac{100}{103}} = \frac{3}{100}$$

Therefore, $S = \frac{1}{2}\cdot\frac{3}{100} = \frac{3}{200}$ To express this as a terminating decimal, we divide 3 by 200: $\frac{3}{200} = \frac{3}{2\cdot 100} = \frac{3}{2\cdot 10^2} = \frac{3}{2}\cdot\frac{1}{100} = 1.5\cdot 0.01 = 0.015$ Alternatively, we can directly divide 3 by 200: $\frac{3}{200} = 0.015$ Thus, the infinite series is equal to $\boxed{0.015}$

Figure I.9: In-place feedback example for the MATH-hard dataset.

In-place feedback example - MATH-hard

Problem
Dr. Zaius invests $\$10,000$ in a CD with an annual interest rate of 4% that compounds semi-annually (twice a year). After six months, he rolls over the CD into another CD with an annual interest rate of 5% that also compounds semi-annually. After six months in the second CD, how much does Dr. Zaius have, in dollars?

Model response
First, we need to calculate the amount after the first six months with the first CD. The principal is 10000, the annual interest rate is 4%, and the interest compounds semi-annually. So the interest rate per period is $4\%/2 = 2\% = 0.02$. The number of periods is $6/2 = 3$. The formula for compound interest is $A = P(1+r)^n$, where A is the amount after n periods, P is the principal, and r is the interest rate per period. In this case, $P = 10000$, $r = 0.02$, and $n = 3$. So, $A = 10000(1+0.02)^3 = 10000(1.02)^3 = 10000(1.061208) = 10612.08$. After six months, Dr. Zaius rolls over the CD into another CD with an annual interest rate of 5% that also compounds semi-annually. The principal is 10612.08. The annual interest rate is 5%, so the interest rate per period is $5\%/2 = 2.5\% = 0.025$. The number of periods is $6/2 = 3$. Using the same formula, $A = P(1+r)^n$, we have $P = 10612.08$, $r = 0.025$, and $n = 3$. So, $A = 10612.08(1+0.025)^3 = 10612.08(1.025)^3 = 10612.08(1.076890625) \approx 11418.53$. Therefore, after six months in the second CD, Dr. Zaius has approximately $\boxed{11418.53}$

1) In-place edit
First, we need to calculate the amount after the first six months with the first CD. The principal is 10000, the annual interest rate is 4%, and the interest compounds semi-annually. So the interest rate per period is $4\%/2 = 2\% = 0.02$. The number of periods is 1.

2) Continuation generation
The amount after six months is $P(1+r)^n = 10000(1+0.02)^1 = 10000(1.02) = 10200$. Then, Dr. Zaius rolls over the CD into another CD with an annual interest rate of 5% that also compounds semi-annually. The principal is 10200, the annual interest rate is 5%, and the interest rate per period is $5\%/2 = 2.5\% = 0.025$. The number of periods is 1. The amount after six months in the second CD is $10200(1+0.025)^1 = 10200(1.025) = 10200 + 10200 \times 0.025 = 10200 + 255 = 10455$. So, after six months in the second CD, Dr. Zaius has $\boxed{10455}$

Figure I.10: In-place feedback example for the MATH-hard dataset.

In-place feedback example - GPQA free-form

Problem
Consider an isolated system of 13 identical particles each with charge 2e and negligible mass. 12 of these charges are constrained to stay at 2 m from a point P. The 13th charge is fixed at P. What is the minimum energy (in Joules) of this system correct to three decimals?

Model response
Let q be the charge of each particle, and r be the distance of the 12 particles from the point P. The 13th charge is at the point P. The Coulomb force between two charges q_1 and q_2 separated by a distance r is given by $F = \frac{kq_1q_2}{r^2}$, where $k = \frac{1}{4\pi\epsilon_0}$ is the Coulomb constant. *[omitted for brevity...]*

The potential energy of the system is given by

$$U = \sum_{i=1}^{12} \frac{kq^2}{r_i}$$

where r_i is the distance from the i-th charge to the point P. Since all 12 charges are at a distance of 2 m from P, $r_i = 2$ for all i. $U = 12 \cdot \frac{kq^2}{2} = 6kq^2$ *[omitted for brevity...]*

The total energy is $U = 6kq^2$. $U = 6 \cdot \frac{1}{4\pi\epsilon_0}(2e)^2 = \frac{6 \cdot 4e^2}{4\pi\epsilon_0} = \frac{6e^2}{\pi\epsilon_0} \approx \frac{6(1.602\times10^{-19})^2}{\pi(8.854\times10^{-12})} \approx \frac{6(2.5664\times10^{-38})}{\pi(8.854\times10^{-12})} \approx \frac{15.40\times10^{-38}}{2.783\times10^{-11}} \approx 5.57 \times 10^{-27} J$. The minimum energy is approximately $\boxed{5.570e-27\text{J}}$

---

1) In-place edit
Let q be the charge of each particle, and r be the distance of the 12 particles from the point P. The 13th charge is at the point P. The Coulomb force between two charges q_1 and q_2 separated by a distance r is given by $F = \frac{kq_1q_2}{r^2}$, where $k = \frac{1}{4\pi\epsilon_0}$ is the Coulomb constant. *[omitted for brevity...]*
The potential energy of the system is given by

$$U = \sum_{i<j} \frac{kq_iq_j}{r_{ij}}$$

---

2) Continuation generation
which includes the 12 central–outer terms and the 66 outer–outer pairwise terms; the outer–outer distances r_{ij} depend on the arrangement of the 12 charges on the sphere of radius 2, so one must evaluate this pairwise sum for the energy-minimizing configuration before computing any numeric value. *[omitted for brevity...]*
The minimum energy is $U = \frac{13}{2}\frac{kq^2}{r} = \frac{13}{2}\frac{\frac{1}{4\pi\epsilon_0}(2e)^2}{2} = \frac{13e^2}{4\pi\epsilon_0} \approx \frac{13(1.602\times10^{-19})^2}{4\pi(8.854\times10^{-12})} \approx 2.82 \times 10^{-26} J$. $\boxed{2.820e-26\text{J}}$

Figure I.11: In-place feedback example for the GPQA free-form dataset.

In-place feedback example - MMLU-pro free-form

Problem
A particle of mass m and velocity u_1 makes a head-on collision with another particle of mass $2m$ at rest. If the coefficient of restitution is such to make the loss of total kinetic energy a maximum, what are the velocities v_1 after the collision?

Model response
Let $m_1 = m$ and $v_1 = u_1$ be the mass and initial velocity of the first particle. Let $m_2 = 2m$ and $v_2 = 0$ be the mass and initial velocity of the second particle. Let v_1' and v_2' be the velocities of the first and second particles after the collision, respectively. The initial total kinetic energy is $K_i = \frac{1}{2}m_1u_1^2 + \frac{1}{2}m_2(0)^2 = \frac{1}{2}mu_1^2$. The final total kinetic energy is $K_f = \frac{1}{2}m_1v_1' + \frac{1}{2}m_2v_2'$.The loss of kinetic energy is $\Delta K = K_i - K_f = \frac{1}{2}mu_1^2 - \frac{1}{2}mv_1' - \frac{1}{2}(2m)v_2'$. *[omitted for brevity...]*

To maximize ΔK, we need to minimize v_2'. Since $v_2' = \frac{1}{2}v_1'$, we have $v_2' = \frac{1}{2}v_1'$. Then $v_1' = 2v_2'$. $\Delta K = \frac{1}{2}mu_1^2 - 2m\frac{1}{2}v_1' = \frac{1}{2}mu_1^2 - mv_1'$. Final Answer: The final answer is $\boxed{0}$

---

1) In-place edit
Let $m_1 = m$ and $v_1 = u_1$ be the mass and initial velocity of the first particle. Let $m_2 = 2m$ and $v_2 = 0$ be the mass and initial velocity of the second particle. Let v_1' and v_2' be the velocities of the first and second particles after the collision, respectively. The initial total kinetic energy is $K_i = \frac{1}{2}m_1u_1^2 + \frac{1}{2}m_2(0)^2 = \frac{1}{2}mu_1^2$. The final total kinetic energy is $K_f = \frac{1}{2}m_1v_1'^2 + \frac{1}{2}m_2v_2'^2$.

---

2) Continuation generation
The loss of kinetic energy is $\Delta K = K_i - K_f = \frac{1}{2}mu_1^2 - \frac{1}{2}m_1v_1'^2 - \frac{1}{2}m_2v_2'^2$. We want to maximize ΔK. The coefficient of restitution is $e = \frac{v_2' - v_1'}{u_1 - v_2'}$. Since it's a head-on collision, we can use the conservation of momentum: $m_1u_1 + m_2(0) = m_1v_1' + m_2v_2'$ $mu_1 = mv_1' + 2mv_2'$ $u_1 = v_1' + 2v_2'$ $v_2' = \frac{u_1 - v_1'}{2}$ *[omitted for brevity...]*

To maximize ΔK, we take the derivative with respect to v_1' and set it to zero. However, we are given that the loss of kinetic energy is maximized. This occurs when ΔK is maximized. For maximum loss of kinetic energy, we want e to be as small as possible, i.e., $e = 0$. If $e = 0$, then $v_1' = \frac{u_1(1-0)}{0+3} = \frac{u_1}{3}$. Then $v_2' = \frac{u_1 - v_1'}{2} = \frac{u_1 - \frac{u_1}{3}}{2} = \frac{\frac{2u_1}{3}}{2} = \frac{u_1}{3}$.
$\boxed{v_1' = \frac{u_1}{3}}$

Figure I.12: In-place feedback example for the MMLU-pro free-form dataset.

Failure example of multi-turn refinement - 1) corrupting correct content

Problem
In convex hexagon $ABCDEF$, all six sides are congruent, $\angle A$ and $\angle D$ are right angles, and $\angle B$, $\angle C$, $\angle E$, and $\angle F$ are congruent. The area of the hexagonal region is $2116(\sqrt{2}+1)$. Find AB.

Model response
To solve for AB in the given convex hexagon $ABCDEF$, we will follow these steps:

1. **Identify the properties of the hexagon:**
- All six sides are congruent.
- $\angle A$ and $\angle D$ are right angles.
- $\angle B$, $\angle C$, $\angle E$, and $\angle F$ are congruent.

2. **Determine the measure of the congruent angles:**
- Since the sum of the interior angles of a hexagon is $(6-2) \times 180^\circ = 720^\circ$.
- Let $\angle B = \angle C = \angle E = \angle F = x$.
- We have $\angle A = 90^\circ$ and $\angle D = 90^\circ$.
- Therefore, $4x + 180^\circ = 720^\circ$.
- Solving for x: $4x = 540^\circ \implies x = 135^\circ$

3. **Visualize the hexagon:** - Place the hexagon on a coordinate plane with A at $(0,0)$ and D at $(a,0)$.
- Since $\angle A = 90^\circ$ and $\angle D = 90^\circ$, the hexagon can be divided into a rectangle and two right triangles.

4. **Divide the hexagon into simpler shapes:**
- The hexagon can be divided into a rectangle $ABCD$ and two right triangles ABE and CDF.

Since the problem asks for AB, and we have determined that $AB = s$, the final answer is: $\boxed{23}$

Feedback
Your output is wrong because you assumed without justification that the hexagon **"can be divided into a rectangle and two right triangles" and then treated $ABCD$ as a rectangle (even as a square) with both width and height equal to s; right angles at A and D alone do not imply $AB \parallel CD$ or $BC \parallel AD$, so $ABCD$ need not be a rectangle and the triangle legs need not equal s.**

Fix it by producing a correct partition justified by the given angles (for example, draw and justify specific diagonals such as BF and CE), use the 135° measure of the other four angles to deduce the shapes and side relationships of the pieces, and compute the area from that valid decomposition rather than assuming a rectangle/square.

Model response
Let's re-evaluate the problem step-by-step, ensuring we correctly decompose the hexagon based on the given angles and properties.

1. **Identify the properties of the hexagon:**
- All six sides are congruent.
- $\angle A = 90^\circ$ and $\angle D = 90^\circ$.
- $\angle B = \angle C = \angle E = \angle F = 135^\circ$.

2. **Visualize the hexagon:**
- Place the hexagon on a coordinate plane with A at $(0,0)$ and D at $(s,0)$.

3. **Draw and justify specific diagonals:**
- Draw diagonal BF and CE. These diagonals will help us decompose the hexagon into simpler shapes.

4. **Analyze the resulting shapes:**
- Diagonal BF splits $\angle B$ and $\angle F$ into two angles each: $\angle ABF = \angle FBD = 45^\circ$.
- Similarly, diagonal CE splits $\angle C$ and $\angle E$ into two angles each: $\angle BCE = \angle ECD = 45^\circ$.

Since the problem asks for AB, and we have determined that $AB = s$, the final answer is: $\boxed{23}$

Figure I.13: Failure example of multi-turn refinement on the MATH-hard dataset: corrupting correct content.

Failure example of multi-turn refinement - 2) ignoring feedback

Problem
A container weighs 3.22 lb force when empty. Filled with waterat 60°F the mass of the container and its contents is 1.95 slugs. Find the weight of the water in the container and its volume in cubic feet. Assume density of water= 62.4 lb force/ft^3.

Model response
Let W_c be the weight of the empty container in lb force, which is given as $W_c = 3.22$ lb. Let m be the mass of the container and its contents in slugs, which is given as $m = 1.95$ slugs. Let ρ be the density of water in lb force/ft^3, which is given as $\rho = 62.4$ lb/ft^3. We want to find the weight of the water in the container, W_w, and the volume of the water in the container, V in cubic feet.

First, we find the weight of the container and its contents due to gravity. We know that weight $W = mg$, where m is mass and g is the acceleration due to gravity. We can use the given mass in slugs and the standard value of $g = 32.2$ ft/s^2. $W = m \cdot g = 1.95 \text{ slugs} \cdot 32.2 \text{ ft/s}^2 = 62.19 \text{ lb}$.

The weight of the container and its contents is also the sum of the weight of the empty container and the weight of the water. $W_c + W_w = 62.19$ lb $3.22 + W_w = 62.19$ $W_w = 62.19 - 3.22 = 58.97$ lb.

Now we can find the volume of the water in the container using the density formula: $\rho = \frac{W}{V}$, where W is the weight of the water and V is the volume. $V = \frac{W}{\rho} = \frac{58.97 \text{ lb}}{62.4 \text{ lb/ft}^3} \approx 0.9455 \text{ ft}^3$.

Therefore, the weight of the water is 58.97 lb and the volume of the water is approximately 0.9455 ft^3.

$\boxed{\text{Weight of water} = 58.97 \text{ lb, Volume} = 0.9455 \text{ ft}^3}$

Feedback
Your output is wrong because you miscomputed the total weight: you evaluated $W = m \cdot g$ incorrectly (you used 1.95·32.2 = 62.19), which led to an incorrect water weight. Recompute $W = m \cdot g$ with the correct product, then subtract the empty container weight to get the water weight and divide that by the given density to obtain the volume.

Model response
Let W_c be the weight of the empty container in lb force, which is given as $W_c = 3.22$ lb. Let m be the mass of the container and its contents in slugs, which is given as $m = 1.95$ slugs. Let ρ be the density of water in lb force/ft^3, which is given as $\rho = 62.4$ lb/ft^3. We want to find the weight of the water in the container, W_w, and the volume of the water in the container, V in cubic feet.

First, we find the weight of the container and its contents due to gravity. We know that weight $W = mg$, where m is mass and g is the acceleration due to gravity. We can use the given mass in slugs and the standard value of $g = 32.2$ ft/s^2. $W = m \cdot g = 1.95 \text{ slugs} \cdot 32.2 \text{ ft/s}^2 = 62.19 \text{ lb}$.

The weight of the container and its contents is 62.19 lb. The weight of the empty container is 3.22 lb. The weight of the water is the difference between the total weight and the weight of the empty container: $W_w = 62.19 - 3.22 = 58.97$ lb.

Now we can find the volume of the water in the container using the density formula: $\rho = \frac{W}{V}$, where W is the weight of the water and V is the volume. $V = \frac{W}{\rho} = \frac{58.97 \text{ lb}}{62.4 \text{ lb/ft}^3} \approx 0.9455 \text{ ft}^3$.

Therefore, the weight of the water is 58.97 lb and the volume of the water is approximately 0.9455 ft^3.

$\boxed{\text{Weight of water} = 58.97 \text{ lb, Volume} = 0.9455 \text{ ft}^3}$

Figure I.14: Failure example of multi-turn refinement on the MMLU-pro free-form dataset: ignoring feedback.

Failure example of multi-turn refinement - 3) introducing a new error

Problem
A 125/250-volt three-wire system has a load on the positive side of 500 amp, on the negative side of 450 amp, the neutral current therefore being 50 amp. If each machine of the balancer set has an efficiency of 86 percent, calculate the current for each of these machines and for the main generator.

Model response
Here's a step-by-step solution to the problem:

1. **Calculate the total power in the positive phase:**
$P_{pos} = V_{pos} \times I_{pos} = 125V \times 500A = 62500W$
2. **Calculate the total power in the negative phase:**
$P_{neg} = V_{neg} \times I_{neg} = 125V \times 450A = 56250W$
3. **Calculate the total power in the neutral phase:**
$P_{neutral} = V_{neutral} \times I_{neutral} = 125V \times 50A = 6250W$
4. **Calculate the total system power:**
$P_{total} = P_{pos} + P_{neg} + P_{neutral} = 62500W + 56250W + 6250W = 125000W$
5. **Determine the number of balancer sets:**
The problem states each balancer set has an efficiency of 86%, which implies that there are two balancer sets. *[omitted for brevity...]*

The balancer sets are designed to correct the imbalance. The generator provides 1000A. The imbalance is 500A - 450A = 50A. The balancer sets must provide 50A. Since each set has 86% efficiency, the current each set provides is $I = \frac{50}{0.86} \approx 581.39A$. So, $I_1 = I_2 = 581.39A$.
$\boxed{I_{generator} = 1000, I_{set1} = 581.39, I_{set2} = 581.39}$

Feedback
Your output is wrong because **you infer there are two balancer sets from the 86% efficiency — efficiency does not determine the number of machines, so splitting total power by two is an unjustified non sequitur that corrupts all subsequent calculations.** To fix it, do not assume a quantity: either use the actual number of machines stated in the problem or introduce a variable n, first compute the required balancing current from the load imbalance (or neutral current), then divide that required output among n machines and account for each machine's 86% efficiency by dividing the required output current/power by 0.86 to obtain the input/current for each machine.

Model response
Okay, you are absolutely correct. My apologies for the flawed reasoning. Let's redo the calculation with the correct approach.

1. **Calculate the total power**:
$P_{total} = 62500W + 56250W + 6250W = 125000W$
2. **Calculate the imbalance:**
The imbalance is the difference between the positive and negative currents: $500A - 450A = 50A$. This is the current the balancer sets need to correct.
3. **Determine the number of balancer sets:**
The problem doesn't explicitly state the number of balancer sets. Let's denote the number of balancer sets as n.
4. **Calculate the required current per balancer set:**
Each balancer set needs to provide $50A$.
5. **Calculate the current required from each balancer set, accounting for efficiency:**
Current per set = (Required current) / (Efficiency) = $50A/0.86 = 581.395A$ (approximately)
$\boxed{I_{generator} = 1000, I_{set1} = 58.14, I_{set2} = 58.14}$

Figure I.15: Failure example of multi-turn refinement on the MMLU-pro free-form dataset: introducing a new error.

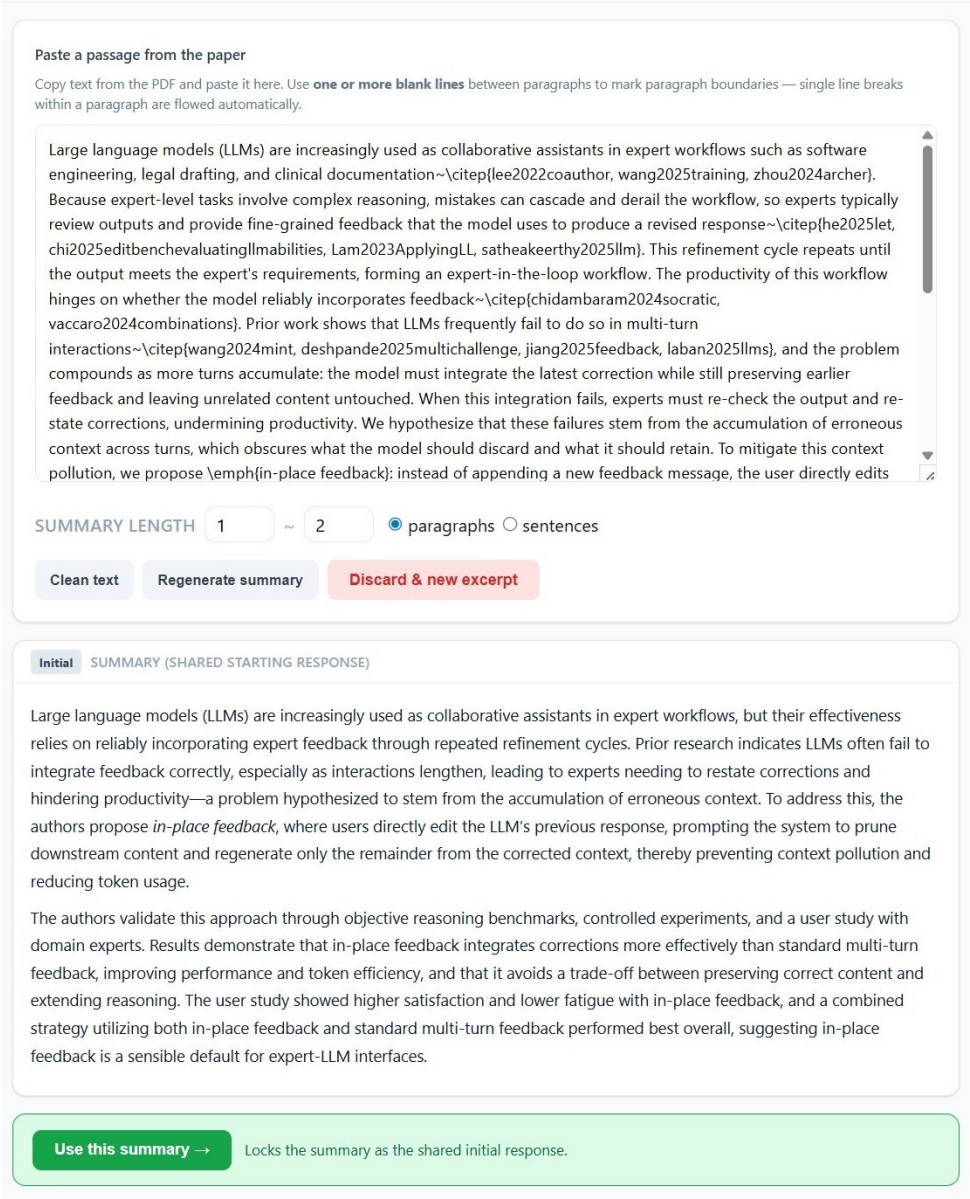

Figure J.1: Screenshot of the summarization page. Participants paste a passage from their chosen paper and click `Summarize` to obtain an initial summary from the model.

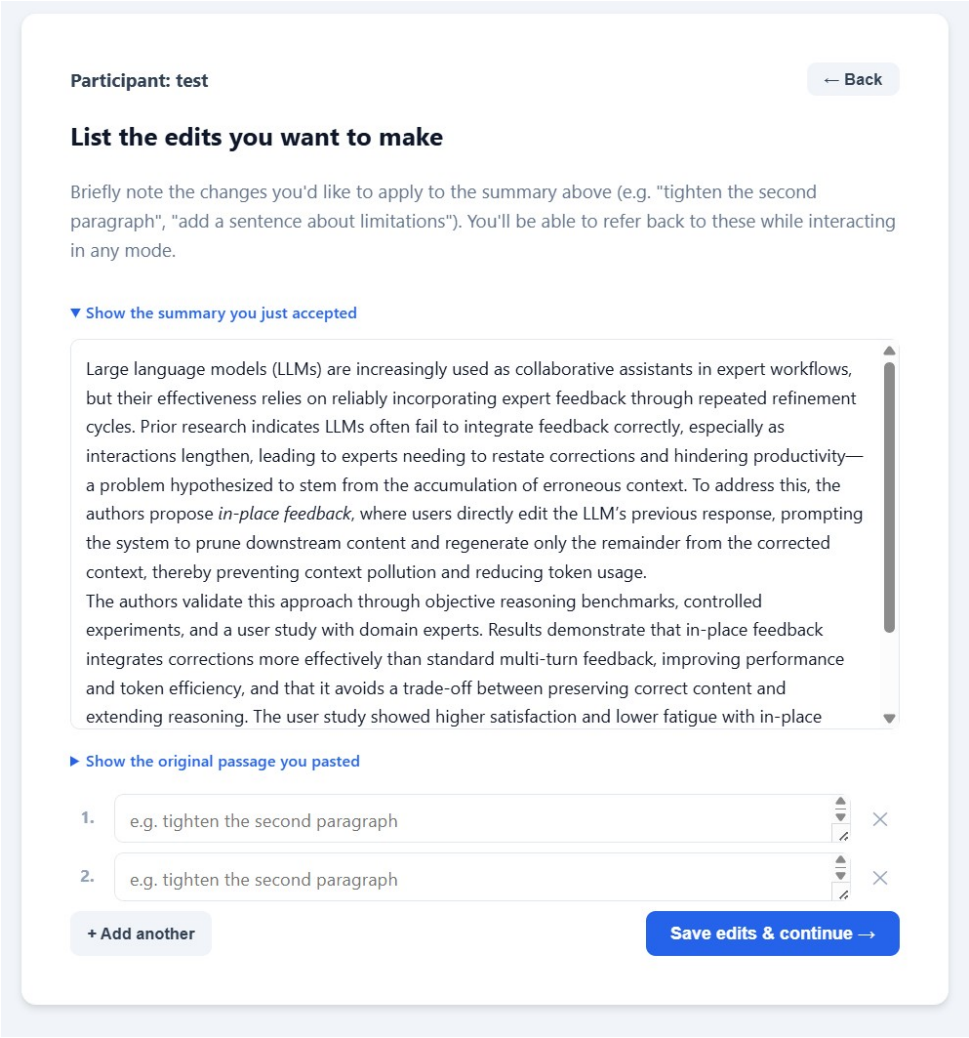

Figure J.2: Screenshot of the edit decision page. Participants record the edits they plan to make. This serves as a memo to help them stay organized during the subsequent interaction.

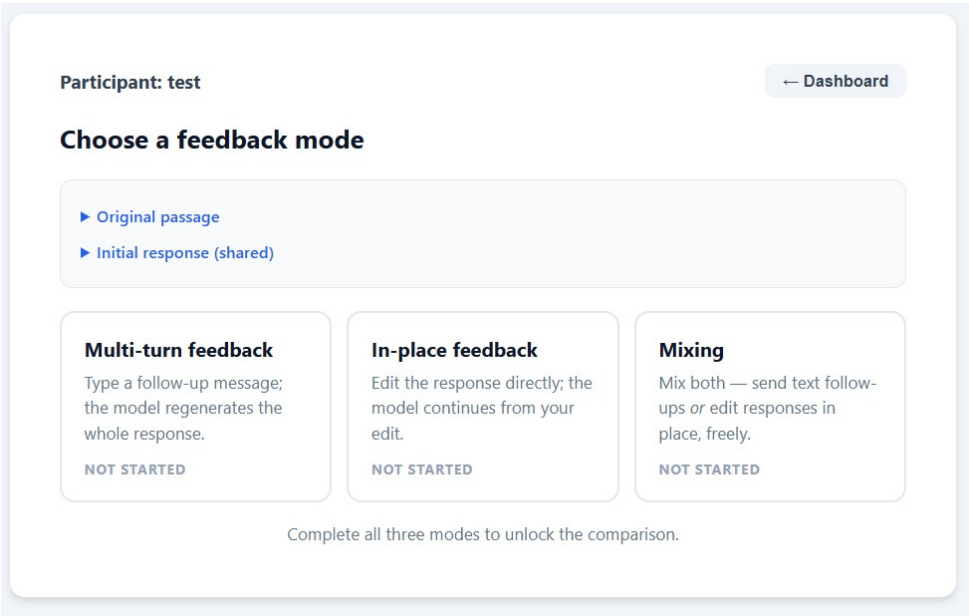

Figure J.3: Screenshot of the mode selection page, where participants choose the editing mode for the upcoming task.

Figure J.4: Screenshot of the multi-turn mode interface, used when participants provide feedback through standard multi-turn feedback.

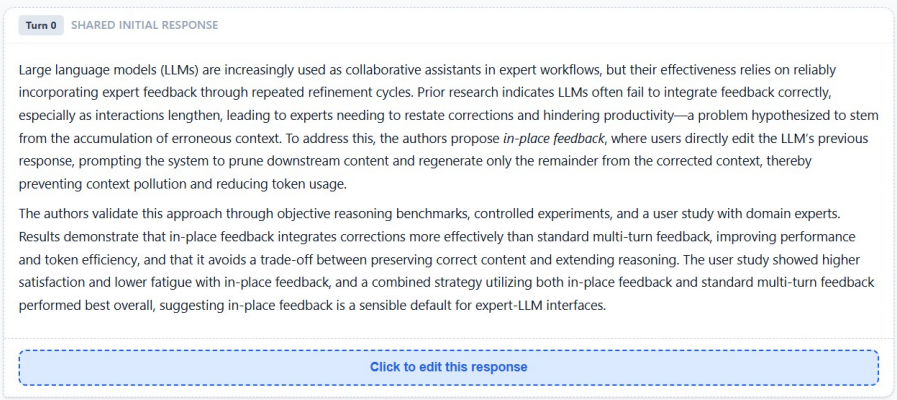

Figure J.5: Screenshot of a model response in the in-place feedback mode. Clicking the button below the response opens an editor that allows participants to perform an in-place edit on the response.

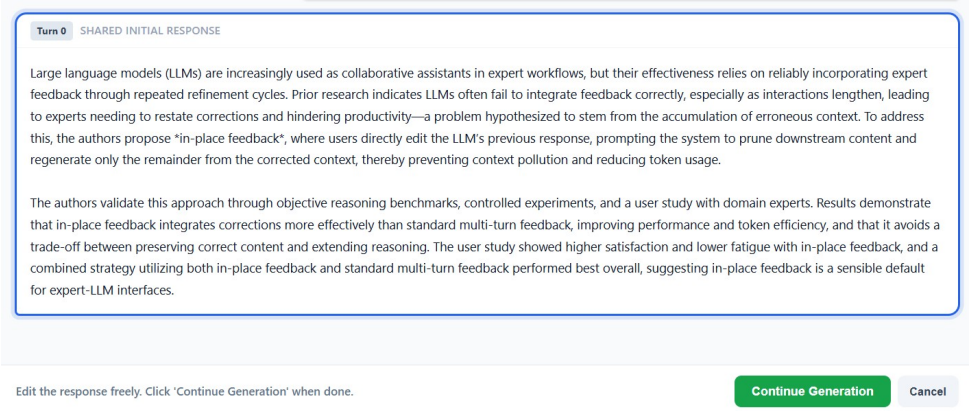

Figure J.6: Screenshot of the in-place feedback interface. Participants directly modify the model's response and click `Continue Generation` to resume generation from the edited context.

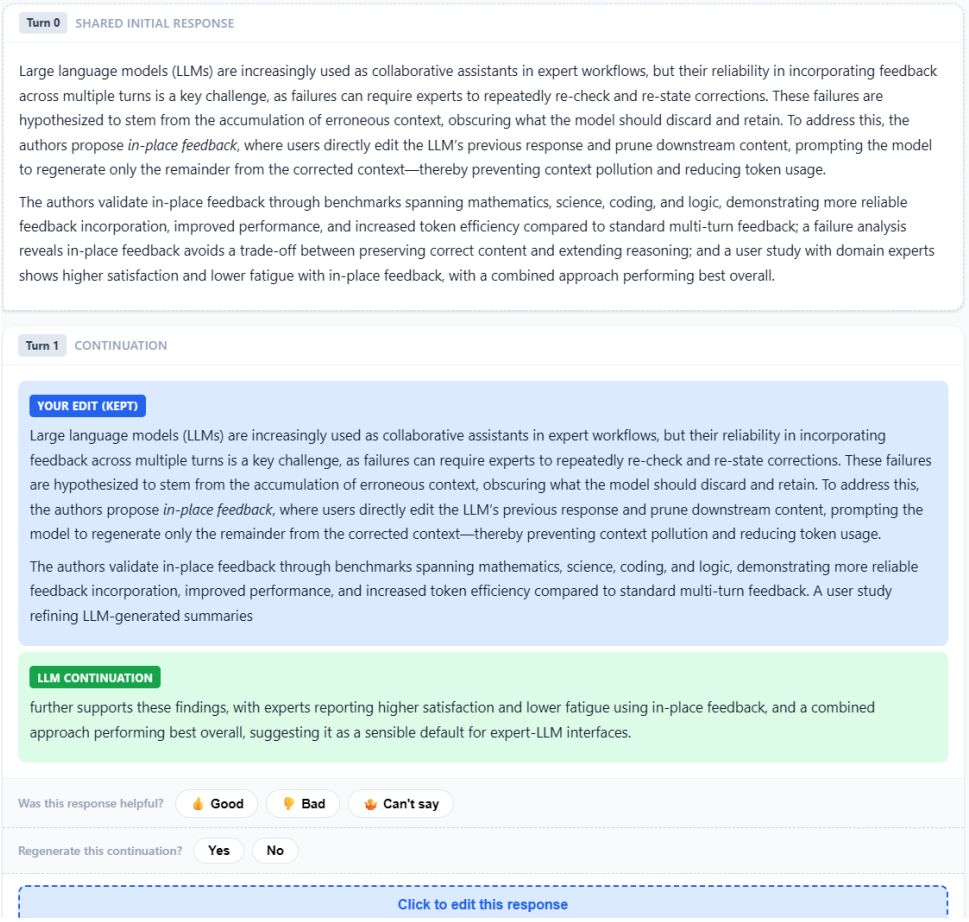

Figure J.7: Screenshot of an example response produced under in-place feedback. The blue span indicates the participant's in-place edit, and the green span indicates the continuation subsequently generated by the model.