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

Anonymous ACL submission

## Abstract

Recent advances in large language models (LLMs) have made it feasible to obtain high-quality first drafts for complex tasks. However, these drafts often contain subtle factual or logical errors. Even with iterative expert feedback, producing an accurate final answer remains difficult. Prior work has shown that LLMs struggle to reliably incorporate multi-turn feedback. In this work, we introduce *in-place feedback*, a novel interaction paradigm in which users directly edit an LLM's previous response. The LLM then conditions on this modified response to generate a revised response. In-place feedback consistently outperforms standard multi-turn feedback across several benchmarks while requiring fewer tokens. Further analysis reveals that in-place feedback applies corrections more reliably and propagates them to subsequent reasoning. Our findings suggest that editing errors directly is a more natural and effective mechanism for LLM-expert collaboration.

## 1 Introduction

Large language models (LLMs) are increasingly used as collaborative assistants in professional workflows, including software engineering, legal drafting, and clinical documentation (Lee et al., 2022; Wang et al., 2025; Zhou et al., 2024). In these settings, the goal is not to replace human expertise, but to help experts work more efficiently. Expert-level tasks often involve complex reasoning, where mistakes can cascade and cause the overall workflow to deviate substantially from the expert's intent. Therefore, experts typically review model outputs and provide targeted feedback to correct errors, after which the model produces a revised response (He et al., 2025; Chi et al., 2025; Lam et al., 2023; Deroy et al., 2024; Satheakeerthy et al., 2025). This refinement cycle can repeat until the output satisfies the target requirements, forming an expert-in-the-loop workflow.

Reliable incorporation of expert feedback is crucial for the success of expert-in-the-loop collaboration. However, prior studies have reported that LLMs do not consistently follow or integrate feedback in multi-turn interactions (Jiang et al., 2025; Laban et al., 2025). This limitation becomes more severe when a task cannot be completed with a single correction and instead requires multiple rounds of feedback. In such cases, the model must (i) retain and respect previously provided feedback and (ii) appropriately integrate newly added feedback, all while preserving unrelated parts of the response. Failure to do so forces experts to repeatedly re-check the output and re-state corrections, undermining productivity.

We hypothesize that these failures stem largely from the accumulation of erroneous context, which can confuse the model regarding which information to discard and which to retain. To mitigate this context pollution, we propose in-place feedback, a novel interaction paradigm. Unlike standard multi-turn feedback, where the user appends a new message to the history, our method allows the user to directly edit the erroneous span within the LLM's previous response. The system then prunes the downstream portion that depends on the edited span and regenerates only the remaining part autoregressively from the corrected context. This design aims to prevent the accumulation of incorrect context across turns. It also keeps the effective context shorter, which can reduce token usage and, consequently, inference cost.

To evaluate the effectiveness of in-place feedback, we conduct experiments on reasoning benchmarks where correctness can be measured objectively: GPQA, MMLU-pro, MATH-hard, and LiveCodeBench. The results show that in-place feedback integrates corrections more effectively than standard multi-turn feedback, leading to stronger task performance. Notably, as the number of refinement turns increases, in-place feedback demon-

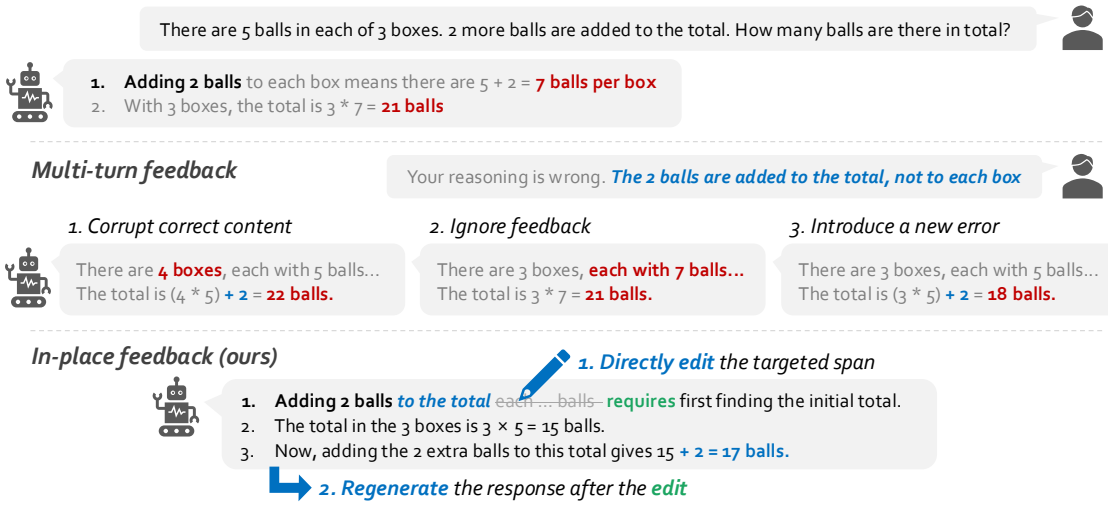



Figure 1: Illustration of common failure cases in multi-turn conversational feedback and in-place feedback. After in-place feedback, the LLM continues generation from the green word "`requires`".

strates larger gains in both performance and efficiency compared to multi-turn feedback.

However, task-level accuracy alone provides limited insight into how in-place feedback changes the model's internal refinement dynamics. To more precisely characterize its effects, we conduct controlled experiments on ZebraLogic (Lin et al., 2025), which enables fine-grained analysis of how each feedback item alters intermediate states and downstream reasoning. Our analysis shows that in-place feedback is incorporated more reliably than multi-turn feedback, especially as turns accumulate. We further find that accumulated dialogue history induces a trade-off: it can help preserve previously correct content, but it can also hinder the model's ability to extend reasoning beyond what is explicitly stated in the feedback. In-place feedback avoids this issue by conditioning directly on the edited context, enabling both dependable feedback acceptance and better propagation of corrections to related reasoning steps.

## 2 In-Place Feedback

### 2.1 Multi-Turn Refinement with Feedback

We first describe how feedback from experts is incorporated into LLMs in interactive settings, focusing on multi-turn refinement. Let $\mathcal{M}$ be an LLM. Given a prompt $x$, the LLM produces an initial response as $y_0 = \mathcal{M}(x)$. An expert then provides feedback to address potential reasoning errors in the initial response. Such feedback can be formalized using a function $\mathcal{F}$, yielding $f_0 = \mathcal{F}(x, y_0)$. In the subsequent turn, the LLM refines its initial response using the feedback and generates the next response conditioned on the problem, the initial response, and the feedback, as $y_1 = \mathcal{M}(x, y_0, f_0)$. This illustrates the refinement process, in which each response is conditioned not only on the prompt but also on the preceding response and its associated feedback.

More generally, the refinement extends to a multi-turn setting, where the LLM iteratively produces responses and incorporates feedback across multiple cycles: $y_t = \mathcal{M}(x, y_0, f_0, y_1, f_1, \ldots, y_{t-1}, f_{t-1})$, 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*.

### 2.2 Motivation: Failure Case 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 Fig. 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 F.2.

Failures of type (i) and (ii) can substantially reduce the efficiency of expert-in-the-loop refinement by forcing experts to repeatedly re-check content or restate 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

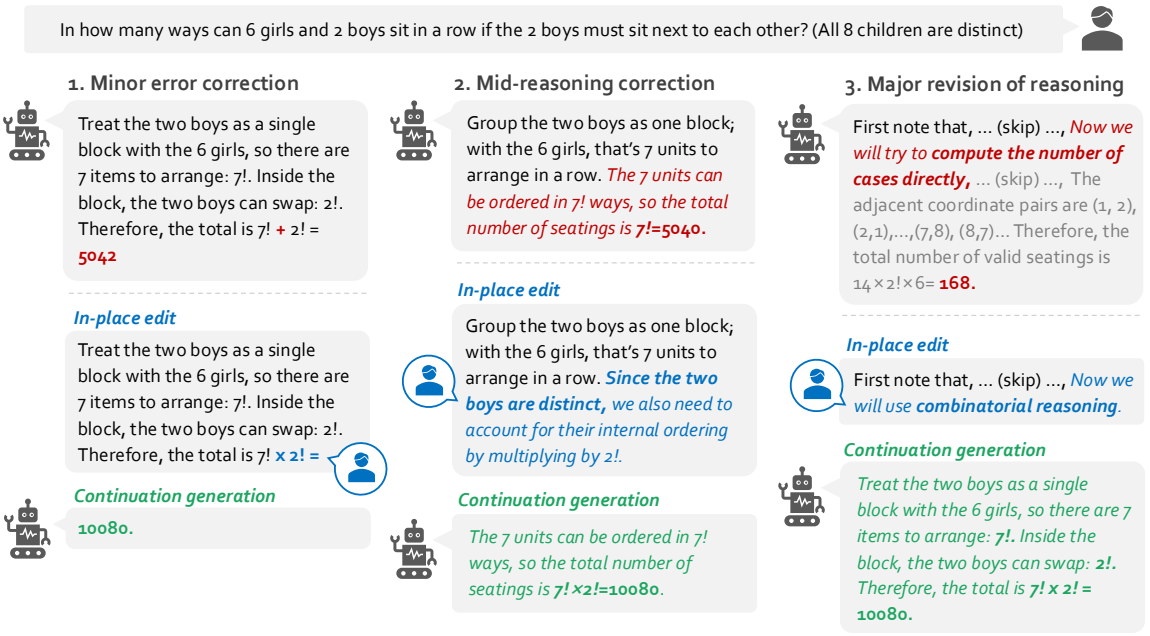



Figure 2: Representative examples of in-place feedback on a toy problem. Red marks incorrect reasoning of the original response, blue indicates the user corrections with in-place feedback, and green shows the subsequent reasoning based on the corrected context. Additional examples are provided in Appendix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

response. We hypothesize that these failures result from the accumulation of incorrect context in the dialogue history, thereby making it difficult for the model to determine which information to retain and which to discard. While failures of type (iii) may appear to stem from the model's reasoning limitations, our analysis in Section 4 reveals that accumulated dialogue history also plays a role.

This hypothesis highlights three requirements for effective refinement. Edits from user feedback should be restricted to the targeted 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?*

### 2.3 In-Place Feedback

To address this question, we propose *in-place feedback*, a new multi-turn interaction paradigm that treats feedback as a state repair rather than a new instruction. As illustrated in Fig. 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 reasoning 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 reasoning 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 reasoning from the correction.

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

**Benefits of in-place feedback.** Multi-turn feedback retains the full interaction history, which allows earlier errors to remain in the context and influence the generation of subsequent responses. In contrast, in-place feedback edits incorrect content directly and retains only the verified spans as the context for subsequent generation. As a result, in-place feedback enables user feedback to directly reflect the generation context and prevents prior errors from influencing subsequent refinement.

In-place feedback also benefits from token efficiency. Multi-turn feedback accumulates a lengthy dialogue history and leads the model to regenerate entire content, including parts that are already correct. In contrast, in-place feedback keeps the history compact by editing only the targeted span and continues generation from the corrected span, avoiding unnecessary regeneration. As a result, it can reduces both input and output tokens.

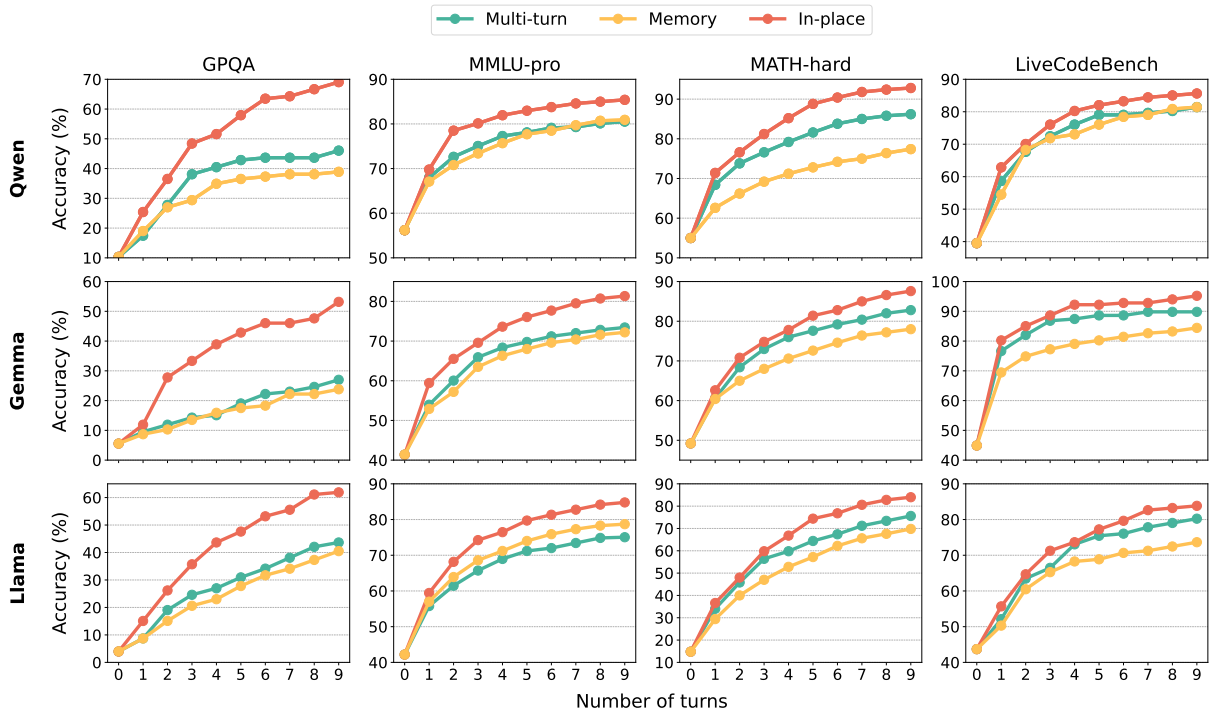



Figure 3: Performance comparison in MATH-hard, MMLU-pro, GPQA, and LiveCodeBench. Across all datasets and LLM models, in-place feedback consistently outperforms the multi-turn and memory-based feedback approaches.

## 3 Empirical Experiments

### 3.1 Experimental Setup

**Datasets and evaluation.** In this section, we evaluate the effectiveness of in-place feedback in multi-turn reasoning. We conduct experiments on three reasoning tasks, MATH-hard (Hendrycks et al., 2021), MMLU-pro (Wang et al., 2024), and GPQA (Rein et al., 2024), and one coding task, Livecodebench (Jain et al., 2024). For MATH-hard, we sample 500 level-5 problems from MATH. For MMLU-pro and GPQA, we use the free-form subsets introduced by Chandak et al. (2025), which contain only open-ended questions. We use code-generation problems in Livecodebench. Details on experimental settings are provided in Appendix A.

**Baselines.** We compare the performance of in-place feedback with multi-turn and memory-based feedback. *Multi-turn feedback* refers to the general interaction paradigm described in Section 2.1. Inspired by prior work that avoids retaining the full interaction history (Shinn et al., 2023; Packer et al., 2023; Madaan et al., 2022), we include *memory-based feedback* as a baseline that conditions only on a limited context. Specifically, at turn $t$, the model refines the previous response $y_{t-1}$ using at most the $k$ recent feedback items: $y_t = \mathcal{M}(P(x, y_{t-1}, f_{t-1}, \ldots, f_{t-k}))$, where $P$ is a prompt template and $k$ is the number of feedback items retained. We use $k = 3$ in our experiments.

**Automated evaluation via LLM Agents.** We replace human experts who generate feedback with LLM-based agents for scalable evaluation. Given a problem, its ground-truth solution, and a LLM's response, the agents identify the earliest critical error and generate feedback.

For in-place feedback, an expert would apply the feedback directly to the previous response. To automate this process, we also use an LLM agent. The agent takes the feedback and the LLM's response, identifies the sentence to be corrected, and provides a replacement. We then substitute the sentence and remove all subsequent text, since it may depend on the corrected span. We use GPT-5-mini (OpenAI, 2025) for both the feedback and in-place agent. Further details on post-processing steps and prompt templates are provided in Appendix A.

**LLMs and hyperparameters.** We use three LLMs: `Qwen2.5-7B-Instruct` (Qwen Team, 2025), `Gemma-3-4b-it` (Gemma Team, 2025), and `Llama-3.1-8B-Instruct` (Kassianik et al., 2025). To examine scalability, we evaluate larger models: `Gemma-3-27b-it` and `Llama-3.1-70B-Instruct`. We run 10 turns for the smaller models and five turns for the larger models. Further experimental settings are provided in Appendix A.

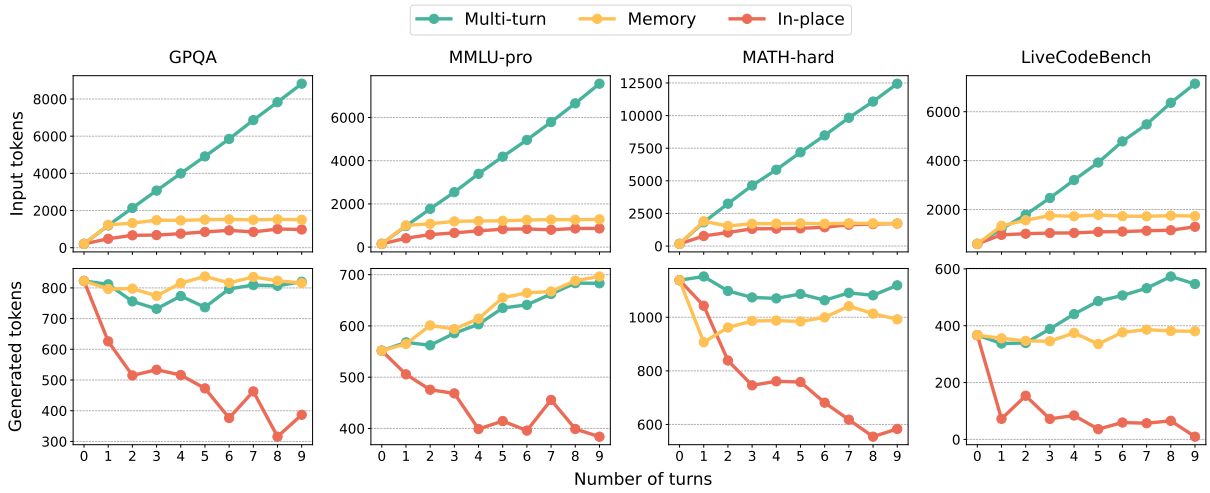



Figure 4: The number of input and generated tokens across multiple turns for `Gemma-3-4b-it`. In-place feedback consistently requires fewer tokens than multi-turn feedback across all datasets and LLMs. The token counts for the other models can be found in Appendix E.

### 3.2 Results

**Task performance.** Fig. 3 shows how accuracy changes as the number of turns increases. Across all 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% 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. These results demonstrate that in-place feedback is a more effective way to integrate external corrections into the reasoning of LLMs. We additionally report results with larger-scale LLMs in Appendix D, showing the same performance trends. We also provide qualitative examples in Appendix F.1 and tabulated results in Appendix E for reference.

**Token efficiency.** Beyond task performance, in-place feedback also exhibits higher token efficiency. Fig. 4 presents the number of input and generated tokens across datasets. For input tokens, the multi-turn feedback appends new turns to the dialogue history, resulting in linear growth with the number of turns. Both memory-based and in-place feedback avoid this accumulation, keeping input length stable across turns. However, the key difference emerges in generated tokens. Memory-based feedback still regenerates the entire response from scratch, similar to multi-turn feedback. In contrast, in-place feedback continues to generate from the corrected span, preserving correct reasoning and producing substantially shorter outputs.

## 4 Fine-Grained Analysis of Turn-Level Refinement Dynamics

Prior work typically evaluates multi-turn interactions solely by checking whether the final output is better than the initial response after all feedback has been applied (Jiang et al., 2025; Sirdeshmukh et al., 2025). Such task-level evaluation leaves open *how feedback actually influences the reasoning process across turns*. Without analyzing turn-level dynamics, it is unclear whether models are using feedback or regenerating new responses. To address this gap, we design controlled experiments with ZebraLogic (Lin et al., 2025), which can generate feedback in a rule-based manner. Using this setting, we compare how multi-turn and in-place feedback incorporate corrections over turns and highlight where in-place feedback provides advantages.

### 4.1 Setup for Controlled Experiments

**Task.** We conduct experiments on the ZebraLogic (Lin et al., 2025), a collection of 573 logic grid puzzles designed to evaluate the reasoning capability of LLMs. Each puzzle consists of $N$ houses and $M$ attributes such as *Name*, *Drink*, and *Hobby*, forming an $N \times M$ grid of cells. Attributes must take $N$ distinct values under uniqueness constraints, resulting in each cell having a single correct value. A set of natural-language clues specifies additional logical relations, and the task is to assign values to all cells so that all constraints are satisfied. Details of the dataset are in Appendix A.2.

**Feedback and in-place agent.** ZebraLogic enables logically grounded feedback generation. We use a Z3 solver (De Moura and Bjørner, 2008) to

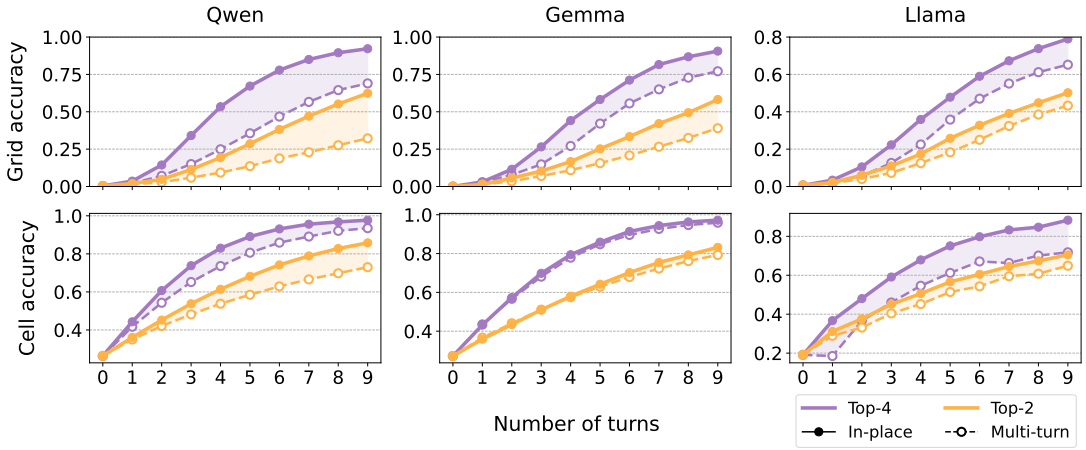



Figure 5: Grid and cell accuracy of LLMs on the Zebralogic dataset. Across both top-2 and top-4 feedback settings, in-place feedback consistently outperforms multi-turn feedback.

systematically derive feedback from constraint violations. We then select the top-$k$ most critical feedback items identified by the solver and provide these to the LLMs using a template. For example, if the model predicts `Name of house 2 = Alice` while the ground truth is `Eric`, the feedback specifies `Name of house 2 is Eric, not Alice`.

Although feedback is systematic, in-place feedback additionally requires identifying where in the response the feedback should be applied. To handle this, we use an in-place agent. Given the feedback and the LLM response, the in-place agent identifies the conflicting span, applies a minimal edit, and removes dependent text to avoid error propagation. We use a controlled procedure to prevent answer leakage. Details are provided in Appendix A.2.

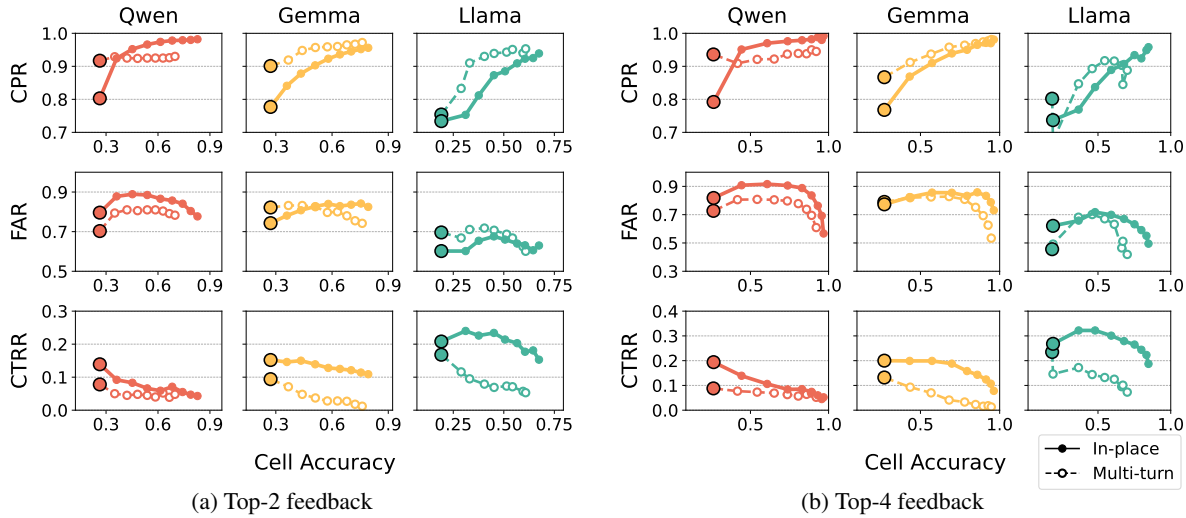
**LLMs and hyperparameters.** We use three open-source LLMs, consistent with the previous experiments. We also conduct experiments on two large-scale LLMs, which are reported in Appendix D. We set $k = 2$ and $k = 4$ for feedback. All experiments are run with three seeds. Detailed experimental settings are provided in Appendix A.

**Metrics.** We evaluate performance using two classes of metrics. To measure overall task performance, we use grid and cell accuracy. Grid accuracy is the fraction of puzzles that are solved perfectly, where all cells match the solution. Cell accuracy is the average fraction of correctly predicted cells across puzzles. To conduct a more fine-grained analysis of the multi-turn refinement, we introduce three complementary metrics:

- **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 that are correct in $y_t$. *This metric evaluates whether the model can retain valid reasoning while applying updates.*
- **Feedback Acceptance Ratio (FAR).** The proportion of cells flagged by feedback $f_t$ that are corrected in $y_{t+1}$, relative to the total number of feedback-provided cells of $y_t$. *This metric captures the model's ability to incorporate explicit corrective signals.*
- **Correction Through Reasoning Ratio (CTRR).** The proportion of cells that are incorrect in $y_t$ but corrected in $y_{t+1}$, relative to the total number of incorrect cells in $y_t$ that are not indicated by feedback $f_t$. *This metric measures the extent to which the model can generalize beyond explicit feedback and improve its reasoning autonomously.*

### 4.2 Analysis of Feedback Utilization

**Task performance.** Fig. 5 shows grid accuracy and cell accuracy as the number of feedback turns increases. In-place feedback consistently outperforms multi-turn feedback, consistent with prior results. Interestingly, the gap in cell accuracy is smaller than the gap in grid accuracy. This suggests that while multi-turn feedback encounters difficulties in correcting the remaining few cells during iterative refinements, in-place feedback is more effective in addressing these corrections. To gain a deeper understanding of this phenomenon, we examine the reasoning dynamics at the turn-level, focusing on how feedback is incorporated and influences the correction process. Larger-scale LLMs show the same trends, as shown in Appendix D.4. We provide additional results with memory-based feedback in Appendix B.2. Since memory-based feedback consistently underperforms multi-turn




(a) Top-2 feedback (b) Top-4 feedback

Figure 6: Correctness-Preserving Rate (CPR), Feedback Acceptance Rate (FAR), and Correction Through Reasoning Ratio (CTRR) for 10-turn conversations of LLMs on the ZebraLogic. 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, we focus on comparing multi-turn and in-place feedback in the main text.

**Turn-level dynamics of LLM behavior with multi-turn feedback.** We first investigate how LLMs behave in multi-turn feedback. Fig. 6 illustrates the dynamics that arise when models repeatedly revise their outputs with feedback. Across all turns, CPR remains above 0.9. `Gemma` and `Llama` show increasing CPR values as the interaction history accumulates, indicating improved retention of previously correct answers in later responses.

However, FAR and CTRR exhibit a steady downward trend over subsequent turns. For FAR, the decline is most evident between turns 5 and 9. CTRR decreases across turns and eventually drops below 0.1. These dynamics suggest that after the first few turns, models become increasingly less receptive to feedback and struggle to reason beyond what is directly specified by that feedback. Notably, `Llama` attains higher CTRR but lower CPR and FAR than the others, indicating a trade-off.

**Advantage of in-place feedback.** We identify two key advantages of in-place feedback compared to multi-turn feedback. First, in-place feedback preserves the ability to incorporate feedback even when the number of turns increases, yielding higher FAR values than multi-turn feedback in later turns. This result accounts for the larger improvement observed in grid accuracy. Second, in-place feedback facilitates reasoning beyond the explicitly targeted errors, thereby leading to consistently higher CTRR than multi-turn feedback. We conjecture that this improvement arises since in-place feedback reduces contextual interference from prior responses, allowing the model to directly condition on the corrected span. By discarding subsequent content and regenerating from the modification, the model may better propagate the corrective signal to related parts of the reasoning, facilitating improvements even in cells not mentioned by the feedback.

The relatively lower CPR and FAR observed for in-place feedback during the earlier turns may reflect the effect of more intensive reasoning compared to multi-turn feedback. Moreover, due to the characteristics of the parallel reasoning problem, an in-place edited span may contain incorrect cell-related reasoning inherited from earlier turns under Top-$k$ feedback, which can propagate errors through subsequent reasoning. In contrast, under oracle feedback, where in-place edits could be error-free, both CPR and FAR are typically higher than in multi-turn feedback, except for CPR with `Qwen` in the first turn (see Fig. A2).

### 4.3 Effect of Dialogue History

A fundamental distinction between multi-turn and in-place feedback lies in the dialogue history. While multi-turn feedback accumulates all prior interactions, in-place feedback conditions generation solely on the edited context. To isolate the impact of dialogue history, we conduct an ablation study with results shown in Fig. 7. Given a multi-turn feedback trajectory, at each turn $t$, we prune the context to retain only the problem, the most recent response, and the immediate feedback

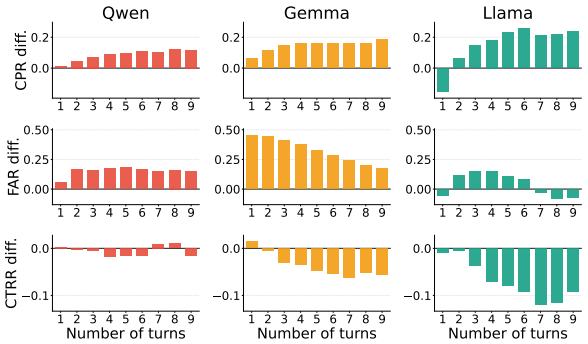



Figure 7: Impact of accumulated dialogue history. For CPR, FAR, and CTRR, we report the difference between the multi-turn feedback rates and the rates under pruned-history settings. Negative values indicate performance degradation due to history pruning.

$(x, y_{t-1}, f_{t-1})$. We then generate a memory-based response $\tilde{y}_t = \mathcal{M}(P(x, y_{t-1}, f_{t-1}))$ and compare it with the original response $y_t$. This comparison isolates the effect of available history on the response at turn $t$. We utilize the Top-4 multi-turn feedback trajectories for this analysis.

**Trade-off between correctness preservation and reasoning.** LLMs exhibit inertia, tending to adhere to their earlier outputs when provided with accumulated history. As shown in Fig. 7, the CPR gap increases with turn depth, indicating that the full history helps models preserve previously correct reasoning. Conversely, the differences in FAR and CTRR under multi-turn feedback decrease as history accumulates. This trend is particularly evident in `Gemma` and `Llama`, suggesting that extensive context history hinders feedback integration and limits the generation of new reasoning steps. We hypothesize that this trade-off is related to induction heads that copy and extend earlier spans (Olsson et al., 2022). We leave a deeper analysis of this mechanism for future work.

**Model-specific patterns on feedback acceptance.** Unlike CPR and CTRR, FAR does not follow a uniform trend across models. `Qwen` consistently achieves higher FAR under multi-turn feedback, whereas `Gemma` shows diminishing gains as turns increase. `Llama` benefits initially, but this advantage 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 heterogeneous patterns to a complex interplay between post-training differences and shared mechanistic inductive biases.

**Complementary feedback mechanisms.** Multi-turn feedback effectively preserves correctness. However, conditioning on accumulated history can impede the model's ability to incorporate new guidance and extend reasoning beyond explicit feedback. In-place feedback mitigates these failures by directly editing the current state. We posit that these paradigms are complementary; they can be employed adaptively based on error type and revision scope to improve both stability and correction quality.

## 5 Related Work

Prior work improves LLMs in multi-turn settings via clarifying questions for ambiguous inputs (Zhang and Choi, 2025; Zamani et al., 2020; Aliannejadi et al., 2019) or via additional training or fine-tuning (Wu et al., 2025; Zhou et al., 2024; Shani et al., 2024). Another stream studies self-refinement, where models critique and revise outputs using internal or external feedback (Madaan et al., 2023; Dhuliawala et al., 2024; Shinn et al., 2023; Nathani et al., 2023; Welleck et al., 2023; Zhang et al., 2025; Huang et al., 2024).

Recent work analyzes LLM behavior in multi-turn conversations, identifying issues such as unreliable feedback incorporation and performance degradation across turns (Jiang et al., 2025; Laban et al., 2025; Sirdeshmukh et al., 2025). However, these studies mainly report high-level metrics and do not examine turn-level dynamics or error propagation. To fill this gap, we conduct a turn-level analysis of feedback-driven refinement and propose a token-efficient interaction scheme that improves multi-turn refinement without additional training.

We provide a more detailed explanation and also discuss the UI-level interaction system, which is closely related to our work, in Appendix C.

## 6 Conclusion

We introduce in-place feedback, an interaction method where experts directly edit an LLM's prior response, and the model generates an output conditioned on this edited context. This approach achieves stronger refinement performance while requiring fewer input and output tokens. Through controlled experiments on ZebraLogic, we show that in-place feedback enables more active interactions than standard multi-turn feedback. While our work focuses on reasoning and coding tasks, we expect in-place feedback to be useful for a wide range of applications, such as document editing and collaborative planning.

## Limitation

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 compute 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.

Although our work focuses on expert–LLM interaction, obtaining experts' judgments for every benchmark instance would be prohibitively costly and difficult to scale. We therefore adopt LLM-based automatic evaluation as a practical and scalable proxy for human assessment in our experiments. This choice allows us to systematically compare different configurations of in-place feedback over a large set of examples, while keeping the evaluation procedure consistent and reproducible.

We analyze turn-level dynamics of the feedback mechanism using ZebraLogic in Section 4. ZebraLogic enables controlled experiments by automatically generating feedback via the Z3 solver, without relying on a feedback agent. While correcting an early step in sequential reasoning tasks would naturally propagate improvements to subsequent reasoning, ZebraLogic involves parallel reasoning where cells can be determined independently, meaning that errors in unedited portions can persist after in-place correction. This issue can be amplified by imperfect edits from the in-place agent, which may fail to remove all dependent reasoning. As a result, our analysis may understate the practical gains of in-place feedback. Nonetheless, the controlled nature of ZebraLogic enables clear attribution of effects, yielding useful insights into refinement dynamics and exposing fundamental constraints of multi-turn feedback.

## Ethical Consideration

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. 2025. Edit-bench: Evaluating llm abilities to perform real-world instructed code edits. *Preprint*, arXiv:2511.04486.

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

Aniket Deroy, Kripabandhu Ghosh, and Saptarshi Ghosh. 2024. Applicability of large language models and generative models for legal case judgement summarization. *Artif. Intell. Law*, 33(4):1007–1050.

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 Annual Meeting of the Association for Computational Linguistics (ACL-Findings)*.

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 (ICLR)*.

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, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

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 *Computer Human Interaction (CHI)*.

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.

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*, pages 1–3.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ved Sirdeshmukh, Kaustubh Deshpande, Johannes Mols, Lifeng Jin, Ed-Yeremai Cardona, Dean Lee, Jeremy Kritz, Willow 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)*.

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 Annual Meeting of the Association for Computational Linguistics (ACL-Findings)*.

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

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 Empirical Study

**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. Both datasets originate from multiple-choice benchmarks but are adapted for generative evaluation via answer matching.

**Motivation for free-form evaluation.** Multiple-choice evaluation is efficient, but it 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. By removing the choices and requiring free-form responses, 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, 493 questions are manually filtered. This dataset offers comprehensive coverage across various domains, making it an ideal resource for evaluating general knowledge and reasoning skills.

The GPQA-Diamond dataset comprises 198 graduate-level science questions, designed to be challenging and test in-depth knowledge and critical reasoning. Similar to 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 high-quality 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 $3 \times 4$ and $3 \times 5$, 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 Fig. A16. 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 prediction, we employ a fuzz score from the Python rapidfuzz library to perform exact-matching evaluation. 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 Fig. A17. 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 the 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 Fig. A18.

### 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. We prevent answer leakage, following Jiang et al. (2025). To prevent leakage, we use the prompts in Fig. A13, Fig. A14, and Fig. A15 and apply post-processing for each agents: (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 Fig. A18. 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 Hyperparameters

All models use temperature 0 for experiments. We observed that `Llama` family can be overly verbose, resulting in degradation of generation quality. To stabilize decoding, we set its repetition penalty to 1.15 for `Llama`. 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.5 Memory-based Feedback

Following Shinn et al. (2023), we retain at most the three most recent feedback items. Fig. A11 shows the prompt template for memory-based feedback.

# B Extended ZebraLogic Results

## B.1 Performance under Oracle Feedback

In addition to top-$k$ feedback, we conduct an additional experiment using oracle feedback, where all available feedback is provided. Fig. A1 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. Fig. A2 illustrates LLM behavior under oracle feedback. In most cases, in-place feedback achieves higher CPR and FAR compared to multi-turn feedback, without CPR of the Qwen in the first turn.

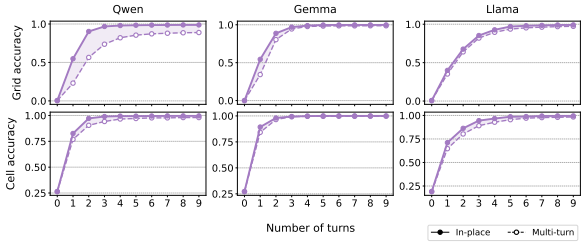



Figure A1: Grid and cell accuracy of LLMs on the Zebralogic dataset. In-place feedback consistently outperforms multi-turn feedback under an oracle setting.

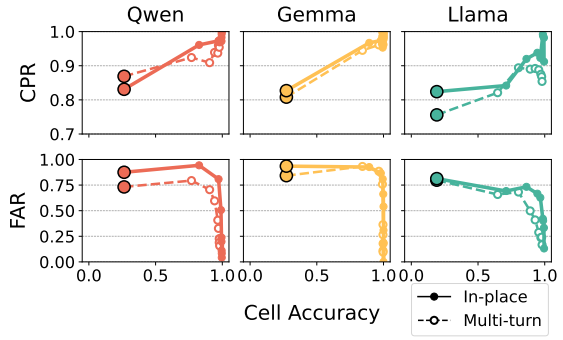



Figure A2: CPR, FAR, and CTRR for 10-turn conversations of LLMs on the ZebraLogic. The Oracle feedback function is used. The points with a black border represent the second response of the LLMs (i.e., $y_1$), and the subsequent responses across turns are connected by lines.

## B.2 Performance of Memory-based Feedback

We run additional experiments with memory-based feedback. As shown in Fig. A3 and Fig. A4, memory-based feedback performs the worst among the three methods. Fig. A5 and Fig. A6 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 4.3 further indicate that the accumulation of dialogue history can contribute to the decreases in FAR and CTRR observed under multi-turn feedback.

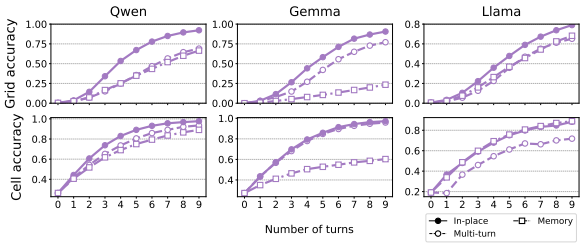



Figure A3: Grid and cell accuracy on the Zebralogic dataset. Top-4 feedback is used.

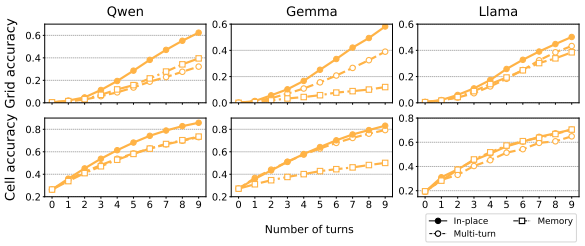



Figure A4: Grid and cell accuracy on the Zebralogic dataset. Top-2 feedback is used.

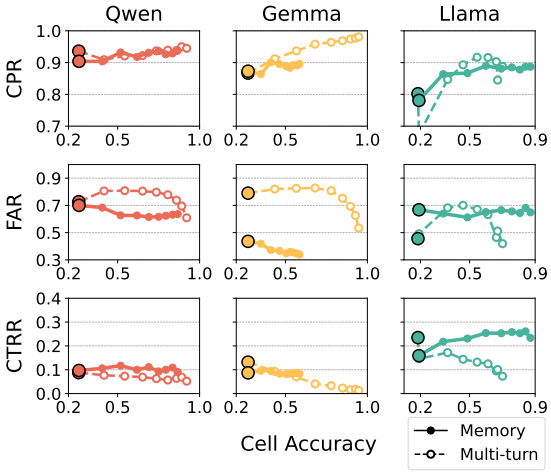



Figure A5: CPR, FAR, and CTRR for 10-turn conversations of LLMs on the ZebraLogic. Top-4 feedback is used. The points with a black border represent the second response of the LLMs (i.e., $y_1$), and the subsequent responses across turns are connected by lines.

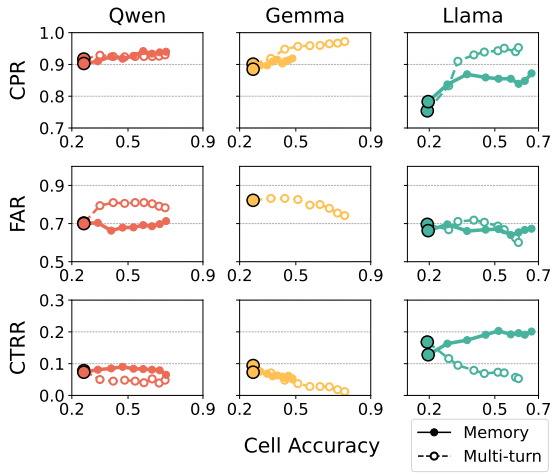



Figure A6: CPR, FAR, and CTRR for 10-turn conversations of LLMs on the ZebraLogic. Top-2 feedback is used. The points with a black border represent the second response of the LLMs (i.e., $y_1$), and the subsequent responses across turns are connected by lines.

## C 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 ambiguous (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 where the 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). However, 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. Han et al. (2025) uses an external LLM agent to provide feedback for evaluating model performance. While these approaches focus on improving feedback quality, we instead analyze turn-level refinement dynamics and propose an alternative interaction paradigm to improve feedback incorporation.

**Analysis on 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. Laban et al. (2025) find that accuracy decreases when a single problem is divided into multiple parts and solved by LLMs in a multi-turn manner. Sirdeshmukh 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.

**UI-level interaction system.** 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 such 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. Motivated by these findings, we extend the idea of localized editing to the model level.

# D Performance of Large-Scale LLMs

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

## D.1 Task Performance

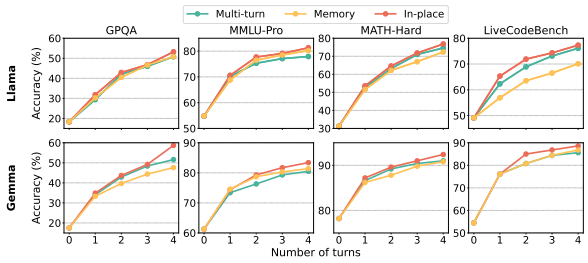



Figure A7: Performance comparison in MATH-hard, MMLU-pro, GPQA, and LiveCodeBench. Across all datasets and large-scale LLM models, in-place feedback consistently outperforms the multi-turn and memory-based feedback approaches. Tabulated results are provided in Table A1, A2, A3, and A4.

Fig. A7 shows the task performance, which follows a similar trend to Fig. 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. Appendix D.2 and Appendix D.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 A1: 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 A2: 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 A3: 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 A4: Performance comparison on LiveCodeBench.

## D.2 Input Token Length

| 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 A5: 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 A6: 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 A7: 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 A8: Number of input tokens on LiveCodeBench.

## D.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 A9: 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 A10: 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 A11: 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 A12: Number of generated tokens on LiveCodeBench.

## D.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.

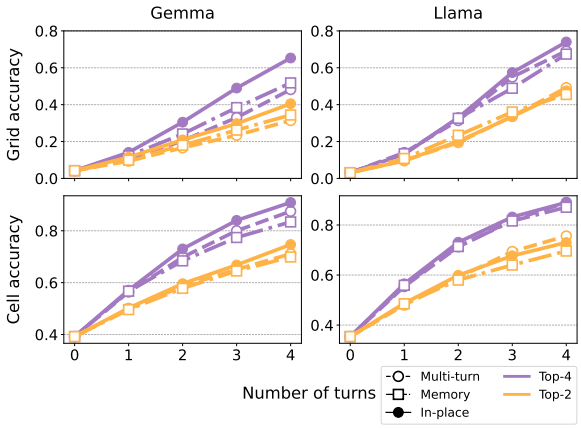



Figure A8: Grid and cell accuracy on the Zebralogic dataset with large-scale LLMs. Tabulated results are provided in Table A13, A15, A14, and A16.

**Task performance.** As shown in Fig. A8, 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.

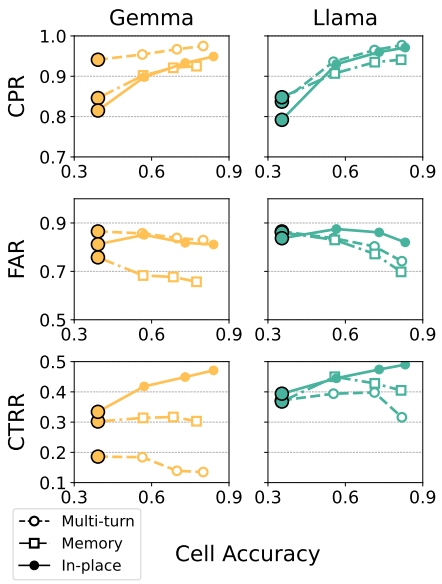



Figure A9: CPR, FAR, and CTRR for five-turn conversations of large-scale LLMs on the ZebraLogic. Top-4 feedback is used. The points with a black border represent the second response of the LLMs (i.e., $y_1$), and the subsequent responses across turns are connected by lines.

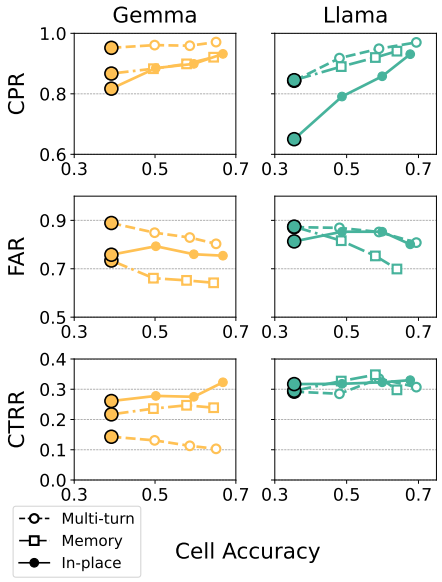



Figure A10: CPR, FAR, and CTRR for five-turn conversations of large-scale LLMs on the ZebraLogic. Top-2 feedback is used. The points with a black border represent the second response of the LLMs (i.e., $y_1$), and the subsequent responses across turns are connected by lines.

**Turn-level dynamics.** Fig. A9 and Fig. A10 illustrate turn-level dynamics for large-scale LLMs. The overall patterns align with Fig. 6. 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 A13: Grid 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 | 0.031 | 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 A14: Grid accuracy of large-scale LLMs on ZebraLogic. Top-2 feedback is used.

| 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 A15: 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 A16: Cell accuracy of large-scale LLMs on ZebraLogic. Top-2 feedback is used.

# E Tabulated Results

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

## E.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 A17: 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 A18: 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 A19: Performance comparison on GPQA.

## E.2 Input Token Length

| | 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 A20: 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 A21: 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 A22: Number of input tokens on GPQA.

## E.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 A23: 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 A24: 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 A25: Number of generated tokens on GPQA.

### E.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 |
| | Inplace | 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 |
| | Inplace | 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 |
| | Inplace | 0.006 | 0.034 | 0.105 | 0.223 | 0.359 | 0.478 | 0.590 | 0.673 | 0.738 | 0.790 |

Table A26: Grid accuracy of LLMs on ZebraLogic. Top-4 feedback is used.

| | 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 |
| | Inplace | 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 |
| | Inplace | 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 |
| | Inplace | 0.006 | 0.021 | 0.060 | 0.110 | 0.174 | 0.258 | 0.328 | 0.391 | 0.448 | 0.502 |

Table A27: Grid accuracy of LLMs on ZebraLogic. Top-2 feedback is used.

| | 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 |
| | Inplace | 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 |
| | Inplace | 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 |
| | Inplace | 0.191 | 0.367 | 0.480 | 0.591 | 0.679 | 0.751 | 0.798 | 0.833 | 0.847 | 0.883 |

Table A28: Cell accuracy of LLMs on ZebraLogic. Top-4 feedback is used.

| | 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 |
| | Inplace | 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 |
| | Inplace | 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 |
| | Inplace | 0.191 | 0.311 | 0.376 | 0.451 | 0.506 | 0.566 | 0.605 | 0.645 | 0.674 | 0.705 |

Table A29: Cell accuracy of LLMs on ZebraLogic. Top-2 feedback is used.

# F Qualitative Examples

## F.1 In-Place Feedback Example

Fig. A19, Fig. A20, Fig. A21, and Fig. A22 are the qualitative examples of in-place feedback on the three benchmarks.

## F.2 Multi-Turn Feedback Failure Example

We observe failure cases of multi-turn feedback, and present the instances in Fig. A23, Fig. A24, and Fig. A25.

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 A11: 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 A12: 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 A13: 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 A14: 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 A15: 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 A16: 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 A17: 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 $\leftrightarrow$ 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 A18: 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.... Then we multiply by $\frac{1}{100}$ to get 0.0001464539.... We can write $\frac{3}{20600} = 0.0001464539...$ The decimal representation is 0.0001464539.... We want to express this as a terminating decimal. We have $S = \frac{3}{20600}$. Dividing 3 by 20600, we get $3 \div 20600 = 0.0001464539...$ 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 $\frac{3}{20600} = 0.0001464539...$ The decimal representation is 0.0001464539.... 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 A19: 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 A20: 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 A21: 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 $\Delta K$, 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 $\Delta K$. 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 $\Delta K$, 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 $\Delta K$ 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 A22: 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 $135^\circ$ 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 A23: Failure example of multi-turn refinement on the MATH-hard free-form 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 $\rho$ 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$^2$. $W = m \cdot g = 1.95$ slugs $\cdot\ 32.2$ ft/s$^2$ $= 62.19$ 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. $V = \frac{W}{\rho} = \frac{58.97 \text{ lb}}{62.4 \text{ lb/ft}^3} \approx 0.9455$ ft$^3$.

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

Weight of water $= 58.97$ lb, Volume $= 0.9455$ 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 $\rho$ 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$^2$. $W = m \cdot g = 1.95$ slugs $\cdot\ 32.2$ ft/s$^2$ $= 62.19$ 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. $V = \frac{W}{\rho} = \frac{58.97 \text{ lb}}{62.4 \text{ lb/ft}^3} \approx 0.9455$ ft$^3$.

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

Weight of water $= 58.97$ lb, Volume $= 0.9455$ ft$^3$

Figure A24: 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 A25: Failure example of multi-turn refinement on the MMLU-pro free-form dataset: introducing a new error.