YOU CHANGED YOUR MIND, THE MODEL DIDN'T: DEMYSTIFYING INTENT IN MULTI-TURN DIALOGUE

Anonymous authors
Paper under double-blind review

ABSTRACT

When a large language model handles a multi-turn task and a user proposes a change but ultimately rejects it, the model should continue as if nothing changed. We find a surprising failure: merely mentioning a rejected change can derail task execution, even when the user's final intent remains unchanged. To systematically study language model behavior under evolving user intent, we introduce `Intent-Eval`, a controlled benchmark spanning tool actions, code, databases, and mathematics. Across diverse tasks, models are vulnerable to both rejected proposals and superseded requirements, consistent with *mentioned-as-in-effect* confusion: conversational content is treated as active requirements even after it has been rejected or replaced. Accuracy degradation can deepen or persist as interaction continues, highlighting the need to distinguish what has been mentioned from what remains in effect. Building on this insight, we propose `Intent-OPSD`, a decision-conditioned on-policy self-distillation framework with Teacher and Student initialized from the same model. The frozen Teacher provides active-intent supervision from the complete task matching the user's decision, training the Student on the full dialogue to follow active requirements reflecting user intent.

1 INTRODUCTION

Large language models complete tasks in multi-turn dialogue, where user intent unfolds gradually. Even with all information provided, spreading a task across turns reduces performance relative to a single message (Laban et al., 2026). Yet users do more than reveal information: they clarify requirements and float changes they later accept or reject. The context then holds rejected or superseded content that should not shape the answer. Recent benchmarks examine requirement changes (Tack et al., 2026; Wu et al., 2026), but do not compare clarifications, accepted changes, and rejected changes on the same tasks. We compare all three and test whether their effects grow or persist in longer dialogues, asking whether models distinguish what was *said* from what remains *in effect*.

Figure 1 illustrates this distinction in a purchase calculation. Once the bag count (4 bags) is introduced, the dialogue splits into four branches: *Original* continues without interruption, while *Neutral* adds clarification without changing what is in effect. In both *Retained* and *Revised*, the user ***mentions*** the same proposal (2 bags), rejecting it in *Retained* and accepting it in *Revised*. Only *Revised* changes what remains ***in effect***, updating the answer from \$300 to \$150. Single-turn controls (Figure 1 (a)) state the same tasks in one message, separating multi-turn failures from task difficulty.

To systematically test whether models follow what remains in effect, we introduce `Intent-Eval`, a benchmark of 414 tasks across Actions, Code, Database, and Math (3,312 evaluation instances). Across eight models, we find three patterns of degradation: ❶ ***Dispersed disclosure:*** splitting the task across turns lowers mean accuracy by 36.30 pp from the single-turn control, even without interruptions (*Original*); ❷ ***Conversational overhead:*** adding neutral clarification without changing the task (*Neutral*) incurs an additional 4.43 pp drop; and ❸ ***Decision-induced confusion:*** a proposed change further degrades performance whether rejected or accepted, including in single-turn controls. In multi-turn dialogue, *Retained* and *Revised* average 8.06 and 5.55 pp below *Original*, respectively.

We further analyze errors and performance in longer dialogues to understand these drops. Accuracy tends to fall as users ask for clarification over more turns or propose and reject more changes. Under *Retained* and *Revised*, new errors relative to *Original* often contain rejected or superseded content, suggesting *mentioned-as-in-effect* confusion: models treat what was said as still in effect.

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

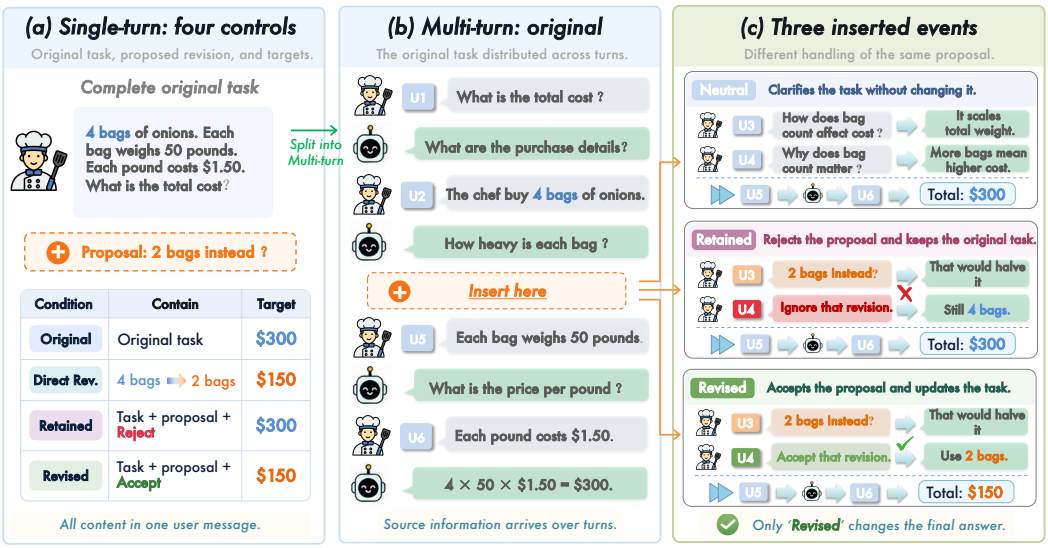


Figure 1: Illustration of active intent. **(a)** Single-turn controls and targets. **(b)** The Original task distributed across turns; U3–U4 mark the insertion point for the other conditions. **(c)** Neutral clarifies; Retained rejects and Revised accepts the same proposal. Only Revised changes the target.

To address this confusion, we use the model's stronger single-turn capability to help it follow what remains in effect in multi-turn dialogue. Notably, the original single-turn task shares active requirements with multi-turn *Original*, *Neutral*, and *Retained*, while the revised single-turn task matches multi-turn *Revised*. This correspondence motivates `Intent-OPSD`, which extends multi-turn self-distillation (Chen et al., 2026b; Zheng et al., 2026) by matching the Teacher to the task that remains in effect after the user's decision. A frozen base model serves as the Teacher and receives the complete single-turn task matching the user's decision, while the Student, initialized from the same model, receives the full dialogue and is trained on-policy to match the Teacher. This focuses supervision on active requirements while the Student learns from the dialogue. At inference, the Student needs neither the Teacher nor the single-turn prompt. Our main contributions are as follows:

- **`Intent-Eval`:** A four-domain benchmark with four controlled multi-turn conditions, *Original*, *Neutral*, *Retained*, and *Revised*, single-turn controls isolating multi-turn effects from task difficulty, and longer dialogues testing whether interference accumulates or persists (Section 2).
- **Empirical findings:** An analysis of accuracy degradation across conditions and how it can deepen or persist as interaction continues. Inactive-content error patterns are consistent with *mentioned-as-in-effect* confusion under *Retained* and *Revised* (Section 4).
- **`Intent-OPSD`:** An on-policy self-distillation method that selects the Teacher's single-turn task based on the user's decision and trains the Student on the full dialogue, improving mean accuracy over the corresponding base models by 10.81 pp across four models and four domains (Section 3).

2 INTENT-EVAL: EVALUATING ACTIVE INTENT

This section describes `Intent-Eval` in detail. We first introduce the four multi-turn conditions (Section 2.1), then explain how these conditions are constructed from source tasks (Section 2.2). Finally, we introduce single-turn controls to separate multi-turn effects from task difficulty and examine how conversational interference changes with interaction depth (Section 2.3).

2.1 MULTI-TURN CONDITIONS

Following Laban et al. (2026), we consider tasks whose requirements are gradually revealed over multiple turns. We split each source task into information shards and reveal them one at a time through user messages u_k, preserving task information while changing only its presentation. The first user message introduces the task goal, and later messages provide the remaining details. After each user message u_k, the assistant generates a response a_k conditioned on the user–assistant history

$$H_k = (u_1, a_1, \ldots, u_{k-1}, a_{k-1}, u_k). \tag{1}$$

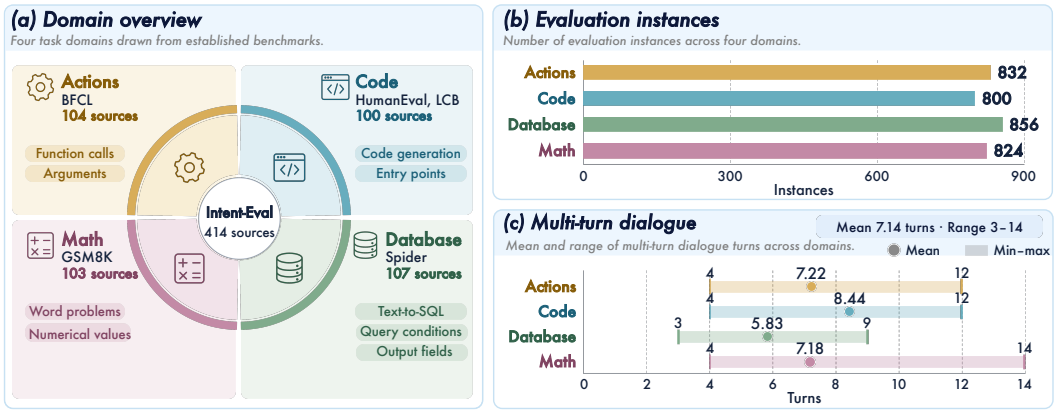


Figure 2: Overview and statistics of `Intent-Eval`. The benchmark comprises 414 source tasks across four domains. Each source task has four multi-turn conditions and four single-turn controls, yielding 3,312 evaluation instances per model.

Beyond revealing task information, users may clarify earlier requirements or propose changes. We call the requirements that remain in effect ***active intent***. To study how models track them, we revise one requirement per source task, with reference answers $y^{(0)}$ and $y^{(1)}$ for the original and revised tasks. We then form four matched multi-turn conditions by continuing disclosure normally or inserting a two-user-message event after the selected requirement is introduced:

- **Original** (M_{Orig})**:** no event is inserted; disclosure continues normally.
- **Neutral** (M_{Neu})**:** two clarifications are inserted; the requirement remains unchanged.
- **Retained** (M_{Ret})**:** a change is proposed and rejected; the original requirement remains active.
- **Revised** (M_{Rev})**:** the same proposed change is accepted; the revised requirement becomes active.

Figure 1 illustrates the protocol. In this example, the selected requirement is introduced at U_2, followed by a two-user-message event at U_3–U_4. *Neutral* adds clarification without proposing a change, while *Retained* and *Revised* use the same proposal, followed by rejection or acceptance at U_4. The dialogue then continues with the remaining task information. Only the final response is scored. *Original*, *Neutral*, and *Retained* are evaluated against the original task requirements and reference answer $y^{(0)}$, while *Revised* is evaluated against the revised requirements and $y^{(1)}$.

2.2 Data and Construction

All 414 source tasks are drawn from the LiC benchmark (Laban et al., 2026), spanning four diverse domains: **Actions** for tool calling from BFCL (Patil et al., 2025), **Code** for program synthesis from HumanEval (Chen et al., 2021) and LiveCodeBench (Jain et al., 2025), **Database** for database querying from Spider (Yu et al., 2018), and **Math** for grade-school math from GSM8K (Cobbe et al., 2021). Figure 2 summarizes the domain distributions, instance counts, and dialogue turn statistics. Each source yields four multi-turn conditions and four single-turn controls, for 3,312 evaluated instances per model. We construct the matched multi-turn conditions in four steps, covering local revisions, reference answers, and condition-specific dialogue events:

I. Candidate enumeration. We apply domain-specific editing rules to the source tasks to generate candidate revisions: changes to tool arguments in **Actions**, entry-point names in **Code**, query conditions or output fields in **Database**, and numerical values in **Math**.

II. Constraint filtering. Each candidate must satisfy three constraints: (1) *Unambiguity*: the change specifies exactly which requirement is replaced and what replaces it; (2) *Atomicity*: the change affects only one requirement while preserving all other requirements; and (3) *Verifiability*: the revised task has a reference answer or output contract that can be checked by the task-specific verifier.

III. Reference answer construction. After filtering, we select and apply one revision per source task, then construct the corresponding reference answer $y^{(1)}$.

IV. Human review and rendering. We review revisions for clarity and consistency with other requirements, then render approved revisions as shared proposals rejected in *Retained* or accepted in *Revised*. *Neutral* adds two questions clarifying the same requirement without proposing changes.

Appendix A provides further construction details and a worked example.

2.3 Controls and Interaction Extensions

Single-turn controls. Our single-turn controls present complete task information in one user message to ❶ separate revised-task difficulty from multi-turn effects; and ❷ examine whether accepted or rejected proposals affect performance without multi-turn interaction. We first define two controls that state the task directly: S_{Orig} presents the original task, while S_{Dir} (Direct Revised) incorporates the revision without a proposal or decision. We then construct two decision-bearing controls by appending the same proposal to S_{Orig}, followed by acceptance or rejection:

$$S_{\text{Rev}} = S_{\text{Orig}} \| [\textsc{Propose}(r'), \textsc{Accept}], \qquad S_{\text{Ret}} = S_{\text{Orig}} \| [\textsc{Propose}(r'), \textsc{Reject}], \tag{2}$$

where r' denotes the proposed replacement requirement and $\|$ denotes concatenation within the same user message. In Figure 1(a), S_{Dir} replaces "4 bags" with "2 bags" in the original prompt. By contrast, S_{Rev} and S_{Ret} retain "4 bags" and append the proposal to use "2 bags", followed by acceptance or rejection, respectively. All other task information remains unchanged. Thus, the original task remains in effect in S_{Orig} and S_{Ret}, which use reference answer $y^{(0)}$; the revised task is in effect in S_{Dir} and S_{Rev}, which use $y^{(1)}$. Accordingly, we match each multi-turn condition to a single-turn control with the same active requirements: $\{M_{\text{Orig}}, M_{\text{Neu}}, M_{\text{Ret}}\} \rightarrow S_{\text{Orig}}$ and $M_{\text{Rev}} \rightarrow S_{\text{Dir}}$. We compare the single-turn controls in two ways:

- **Task difficulty (S_{Orig} vs. S_{Dir}):** Tests whether the revised task is harder when stated directly.
- **Decision effects (S_{Dir} vs. S_{Rev}; S_{Orig} vs. S_{Ret}):** Tests whether expressing the same active task through a proposal and decision affects performance relative to stating it directly.

Extended interaction. We use two protocols to test whether interference accumulates or persists during interaction, with added-block count k from 0 to 4 (Appendix C provides construction details):

- **Accumulation (Neutral $\times$ k vs. Retained $\times$ k):** Starting from *Original* ($k = 0$), we insert k clarification blocks or k proposal-and-rejection blocks about the same requirement. Each rejection block contains a distinct proposal. Both paths preserve the original task and reference answer, allowing us to compare how performance changes as clarification or rejection accumulates.
- **Persistence ($\{$Retained, Revised$\} \rightarrow$ Neutral(k)):** We add k neutral clarification blocks after either decision, keeping the active task unchanged. Comparison with each condition's $k = 0$ baseline tests whether interference persists during clarification and how it affects performance.

3 Intent-OPSD: Learning Active Intent

Building on the active-task matching established in Section 2.3, `Intent-Eval` provides a complete single-turn task for each multi-turn condition, specifying the same active requirements. As shown in Figure 3, `Intent-OPSD` uses this single-turn task for supervision: the Student receives the full dialogue, while a frozen Teacher receives the corresponding complete task prompt.

3.1 Active-Task Supervision and Source Selection

Our goal is to supervise task execution according to requirements that remain in effect, rather than all requirements mentioned in the dialogue. For each multi-turn condition indexed by v, we use its corresponding complete single-turn task as the Teacher prompt:

$$c_v = \begin{cases} S_{\text{Dir}}, & \text{if } v = \text{Rev}, \\ S_{\text{Orig}}, & \text{otherwise.} \end{cases} \tag{3}$$

This selection follows the active task: rejecting a proposal leaves the Teacher on the original task, while accepting it switches the Teacher to the revised task. The Teacher thus provides supervision

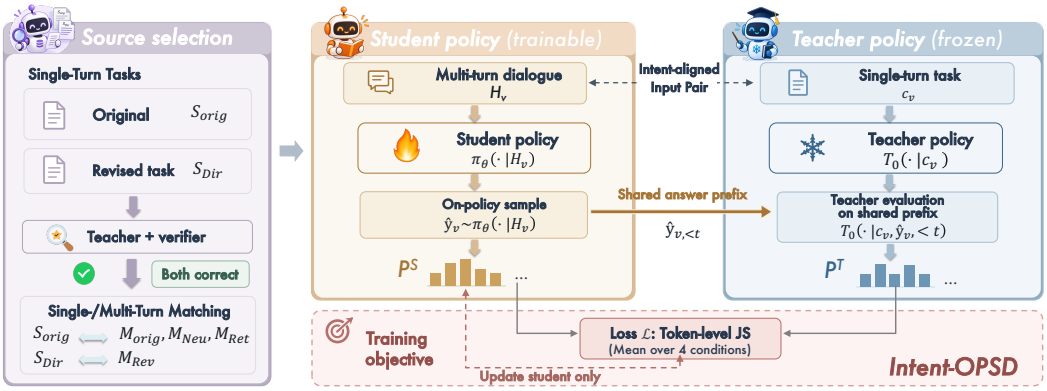


Figure 3: `Intent-OPSD`: the frozen Teacher sees the active task; the Student sees the dialogue.

from a complete statement of the active requirements, guiding the Student to identify and integrate those requirements from the full dialogue despite redundant, rejected, or superseded content.

For each candidate source, we first evaluate the Teacher on both complete single-turn tasks, S_{Orig} and S_{Dir}. We retain a source for training only if both responses terminate normally and pass their respective verifiers. This check confirms that the Teacher can solve both the original and revised tasks from complete single-turn prompts. During training, the Teacher uses these prompts to supervise the Student under the matching multi-turn conditions. Reference answers and verifier results are used only for source selection and are never provided to either model.

3.2 On-Policy Distillation

For each admitted source and condition v, let H_v denote the multi-turn dialogue history under M_v before the final response. The Student receives H_v, while the frozen Teacher receives the corresponding single-turn Teacher prompt c_v. M_{Ret} and M_{Rev} share the dialogue prefix through the Student's response to the same proposal, then branch at the user's decision. The Student then samples the final answer on-policy $\hat{y}_v \sim \pi_\theta(\cdot \mid H_v)$.

At each final-answer position t, both policies evaluate the next-token distribution using the same Student-generated answer prefix $\hat{y}_{v,<t}$:

$$p^S_{v,t} = \pi_\theta(\cdot \mid H_v, \hat{y}_{v,<t}), \qquad p^T_{v,t} = T_0(\cdot \mid c_v, \hat{y}_{v,<t}), \qquad \mathcal{L}_v = \frac{1}{|\hat{y}_v|} \sum_{t=1}^{|\hat{y}_v|} \mathrm{JS}\big(p^S_{v,t}, p^T_{v,t}\big), \tag{4}$$

where T_0 is the frozen base-model Teacher and JS is Jensen–Shannon divergence over the full vocabulary (Lin, 1991). The Teacher scores Student-generated prefixes without generating a separate answer. Intermediate replies remain in H_v but receive no loss.

We then average the four condition losses equally:

$$\mathcal{L} = \frac{1}{4}\left(\mathcal{L}_{\text{Orig}} + \mathcal{L}_{\text{Neu}} + \mathcal{L}_{\text{Ret}} + \mathcal{L}_{\text{Rev}}\right). \tag{5}$$

Only the Student is updated, while the Teacher remains frozen. The updated Student generates subsequent rollouts, maintaining on-policy training. At inference, the Student responds directly from the dialogue without the Teacher or the complete-task prompt.

4 Experiments

In this section, we study accuracy, error patterns, continued interaction, and decision-conditioned distillation through four research questions:

- **RQ1:** How does multi-turn accuracy vary under different matching conditions? ***(Phenomenon)***
- **RQ2:** What characterizes errors under *Retained* and *Revised*? ***(Diagnosis)***
- **RQ3:** Does interference accumulate or persist as the dialogue continues? ***(Depth)***
- **RQ4:** Can decision-conditioned distillation improve multi-turn accuracy? ***(Improvement)***

Table 1: Model-by-domain accuracy (%) across single-turn controls and the four multi-turn conditions. In Single, Direct Rev. states the revised task directly, while Revised includes the proposal and acceptance. Overall aggregates all 414 task sources across the eight models shown here; two additional models appear in Appendix Table 8. Cell backgrounds indicate changes relative to Single Original: for declines, for gains, and for no change.

Model	Domain	Single				Multi			
		Original	Direct Rev.	Revised	Retained	Original	Neutral	Revised	Retained
3.6-27B	Math	98.06	97.09	87.38	93.20	73.79	71.84	66.99	61.17
	Code	95.00	93.00	90.00	88.00	81.00	76.00	80.00	72.00
	DB	92.52	91.59	91.59	87.85	42.99	43.93	42.06	38.32
	Actions	96.15	95.19	96.15	85.58	49.04	44.23	47.12	48.08
3-8B	Math	92.23	90.29	75.73	67.96	64.08	64.08	54.37	37.86
	Code	73.00	75.00	42.00	71.00	41.00	43.00	43.00	26.00
	DB	85.98	85.05	74.77	50.47	38.32	26.17	32.71	17.76
	Actions	97.12	96.15	83.65	85.58	51.92	50.00	46.15	39.42
3.1-8B	Math	75.73	72.82	62.14	64.08	33.98	29.13	22.33	17.48
	Code	45.00	47.00	43.00	24.00	25.00	21.00	16.00	10.00
	DB	79.44	78.50	76.64	43.93	22.43	24.30	16.82	19.63
	Actions	93.27	93.27	72.12	80.77	55.77	52.88	48.08	48.08
4-26B-A4B	Math	97.09	98.06	87.38	92.23	69.90	65.05	66.02	58.25
	Code	92.00	91.00	71.00	89.00	76.00	54.00	68.00	59.00
	DB	93.46	90.65	88.79	91.59	42.06	44.86	42.06	35.51
	Actions	98.08	98.08	92.31	92.31	40.38	31.73	36.54	28.85
5.6-Luna	Math	95.15	96.12	90.29	84.47	76.70	73.79	64.08	62.14
	Code	96.00	92.00	93.00	83.00	93.00	92.00	93.00	85.00
	DB	94.39	93.46	93.46	90.65	42.99	39.25	48.60	36.45
	Actions	86.54	86.54	91.35	88.46	41.35	40.38	34.62	33.65
3.7-Flash	Math	97.09	99.03	92.23	95.15	72.82	66.99	61.17	67.96
	Code	99.00	95.00	83.00	99.00	99.00	68.00	44.00	94.00
	DB	99.07	99.07	95.33	98.13	42.06	43.93	38.32	36.45
	Actions	98.08	98.08	99.04	98.08	37.50	32.69	39.42	41.35
V4-Flash-0731	Math	98.06	97.09	94.17	93.20	56.31	57.28	54.37	54.37
	Code	98.00	99.00	100.00	97.00	95.00	90.00	86.00	93.00
	DB	92.52	93.46	86.92	92.52	36.45	41.12	31.78	35.51
	Actions	94.23	93.27	94.23	94.23	37.50	34.62	30.77	33.65
Sonnet-5	Math	95.15	92.23	87.38	89.32	65.05	67.96	68.93	65.05
	Code	97.00	92.00	97.00	96.00	88.00	58.00	88.00	81.00
	DB	94.39	93.46	93.46	92.52	42.06	45.79	41.12	34.58
	Actions	95.19	93.27	95.19	95.19	50.96	45.19	51.92	53.85
Eight-model overall		91.73	90.85	85.11	84.21	55.43	51.00	49.88	47.37

Experimental settings. We evaluate ten LLMs from six providers: Qwen (Qwen3.6-27B, Qwen3-8B, Qwen3-4B-Instruct-2507, and Qwen2.5-7B-Instruct), Meta's Llama (Llama-3.1-8B), OpenAI (GPT-5.6-Luna), DeepSeek (DeepSeek-V4-Flash-0731), Anthropic (Claude-Sonnet-5), and Google (Gemini-3.7-Flash and Gemma-4-26B-A4B). **RQ1** reports eight models in the main table, with Qwen3-4B and Qwen2.5-7B in the supplementary results. Each model runs four single-turn controls and four multi-turn conditions. Multi-turn runs use a fixed Qwen3.6-27B user simulator, with assistants using dialogue history (see Appendix B.3 for simulator reliability). **RQ2** pairs outputs by source and model across ten models, while **RQ3** varies interaction depth.

For **RQ4**, we compare `Intent-OPSD` with the Base and matched supervised fine-tuning (SFT). The main comparison uses Qwen3-8B, Gemma-4-26B-A4B, Qwen2.5-7B-Instruct, and Llama-3.1-8B. Each domain provides 500 candidate training sources and 50 validation sources, with no overlap with evaluation sources. We train on candidates for which the Teacher solves both task variants correctly. We fully open-source all related data and code[1] for future research. More detailed evaluation and training settings can be found in Appendices B, C, and D.

[1] https://anonymous.4open.science/r/intent-DEC2/

4.1 MULTI-TURN ACCURACY ACROSS CONDITIONS (RQ1)

Overall performance. Table 1 compares single-turn controls and multi-turn conditions across eight models and four domains. We observe three patterns.

(1) The multi-turn gap decomposes into disclosure and conversational overhead. As in prior work, distributing the same task across turns lowers mean accuracy by 36.30 pp. More surprisingly, adding *Neutral* clarification causes a further 4.43 pp drop, even though it changes neither the active task nor the reference answer. This pattern appears across all eight models, suggesting that additional discussion can interfere with task execution even when it introduces no new requirement.

(2) Decision-induced confusion. Introducing a proposed change lowers multi-turn accuracy by 5.55 pp when accepted and 8.06 pp when rejected, relative to *Original*. Compared with *Neutral*, the additional drops are 1.12 and 3.63 pp, respectively. This is not simply because the revised task is harder: models perform similarly when both tasks are stated directly. Even in a single turn, *Revised* and *Retained* reduce accuracy by 5.74 and 7.52 pp, respectively, compared with stating the same active task directly. Thus, models struggle to determine what remains in effect after a decision.

(3) Frontier models show that strong multi-turn robustness is achievable in Code. GPT-5.6 and DeepSeek-V4 retain at least 85% accuracy across all multi-turn Code conditions, contrasting with the larger degradation observed in Database, Actions, and Math. This suggests that recent progress in conversational and agentic coding may improve multi-turn robustness. However, the advantage is highly domain-specific: similarly strong robustness does not emerge in the other structured tasks.

4.2 CHARACTERIZING ERRORS UNDER *Retained* AND *Revised* (RQ2)

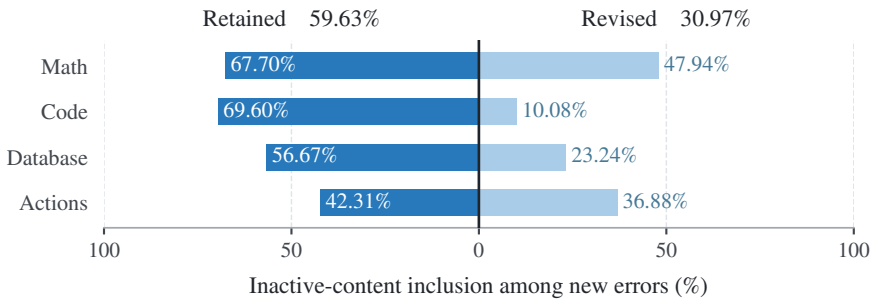


Figure 4: Inactive content in errors. Bars show inclusion rates among new *Retained* (left) and *Revised* (right) errors, pooled over ten models; headers aggregate all domains.

Settings. To diagnose the accuracy gap, we analyze multi-turn cases that succeed in *Original* but fail in *Retained* or *Revised*, checking for inactive content in their final responses. Detection details, per-model results for all ten models, and illustrative examples appear in Appendix E.2.

(1) Errors often include inactive content. Under *Retained*, inactive content is the rejected proposal; under *Revised*, it is the original requirement replaced by the proposal. As shown in Figure 4, inactive content appears in 59.63% of new *Retained* errors and 30.97% of new *Revised* errors across ten models. These results show that inactive content often appears in these erroneous responses, particularly after rejection, a pattern consistent with *mentioned-as-in-effect* confusion.

(2) Math shows relatively high inactive-content inclusion under both decisions. In Math, these rates reach 67.70% under *Retained* and 47.94% under *Revised*. This may reflect the step-by-step nature of mathematical reasoning: changing or restoring a numerical premise can require recomputing several intermediate results. If earlier calculations are reused without being updated, rejected or superseded values may persist in subsequent steps.

4.3 DOES INTERFERENCE ACCUMULATE OR PERSIST? (RQ3)

Settings. Accumulation uses the 372 of 414 sources that have four distinct proposals. Added blocks leave the active requirements and reference answer unchanged. Persistence uses 414 sources, adding *Neutral* clarification after *Retained* or *Revised* without changing the active requirements.

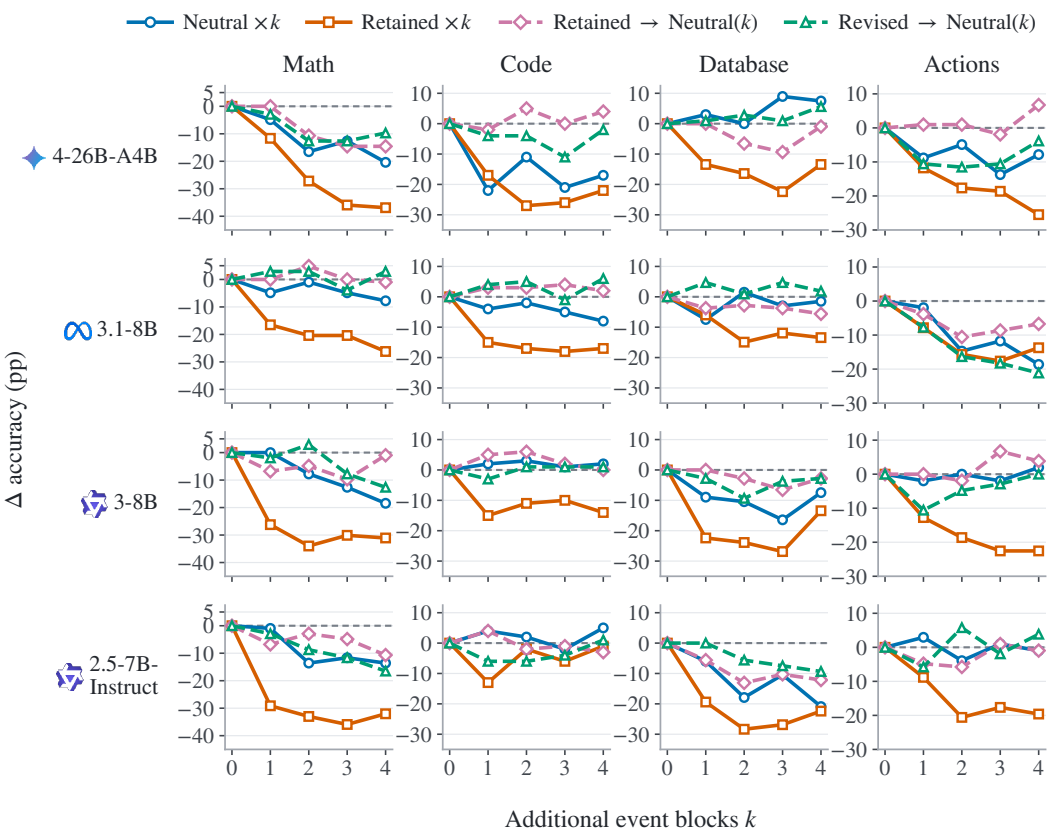


Figure 5: Accuracy change with depth relative to each curve's $k = 0$ baseline. Y-ranges are shared within domains. Results for six additional models are in Appendix E.3 (Figure 7).

(1) Accumulation: the first block causes the largest average drop, with further losses at greater depth. Across four models in Figure 5, the first block produces the largest average single-step decline. Accuracy generally continues to decline as blocks are added. At the final depth, accuracy falls by 8.13 pp for *Neutral* and 20.77 pp for *Retained* relative to Original. As event messages accumulate, repeatedly proposing and rejecting changes produces greater degradation than clarification.

(2) Persistence: continued clarification leaves accuracy below the decision baseline. Relative to each *Retained* or *Revised* baseline, mean accuracy stays lower across all tested clarification depths. Adding four *Neutral* blocks lowers accuracy by 2.72 pp under *Retained* and 3.50 pp under *Revised*. Further discussion can therefore hurt performance even after the user has clearly accepted or rejected the proposal and the active task remains unchanged.

4.4 Can decision-conditioned distillation improve accuracy? (RQ4)

(1) Active-task distillation improves multi-turn accuracy. `Intent-OPSD` improves mean accuracy over Base and SFT by 10.81 and 3.73 pp, respectively (Table 2). It outperforms Base in all 16 model–domain pairs and SFT in 13. Its mean gains over both baselines remain positive across tested depths. Compared with SFT, gains are strong in *Revised*, while mean *Retained* accuracy is similar. Gains over Base suggest the Teacher's active-task supervision helps the Student follow active requirements, even when the dialogue contains rejected or superseded content.

(2) `Intent-OPSD` may help when intent tracking is a bottleneck. `Intent-OPSD` improves Actions accuracy by 21.46 pp over Base, where selecting the correct call arguments can fix the output. Math and Database may similarly benefit from tracking the numerical premises and query constraints that remain active. Code shows smaller gains, which may reflect its demands on program structure and logic. Even when the model correctly tracks the active requirement, structural or implementation errors can still cause the solution to fail. These results suggest that `Intent-OPSD` can be useful when models can solve the task but struggle to track which requirements to follow.

Table 2: Accuracy (%) and `Intent-OPSD` depth gains (pp). For accuracy, conditions and model–domain pairs have equal weight. Colors show changes from Base (lower, higher, unchanged); shading reflects magnitude. Gains use overall accuracy across the four interaction paths; k counts added event blocks. Appendix E.4 provides detailed results by path and depth.

Model	Domain	Base				SFT				`Intent-OPSD`			
		Orig.	Neut.	Revised	Retained	Orig.	Neut.	Revised	Retained	Orig.	Neut.	Revised	Retained
3-8B	Math	64.08	64.08	54.37	37.86	61.17	61.17	59.22	28.16	66.02	67.96	64.08	23.30
	Code	41.00	43.00	43.00	26.00	35.00	36.00	37.00	24.00	39.00	42.00	45.00	28.00
	DB	38.32	26.17	32.71	17.76	43.93	38.32	43.93	20.56	45.79	44.86	42.99	27.10
	Actions	51.92	50.00	46.15	39.42	68.27	71.15	31.73	57.69	66.35	60.58	66.35	55.77
4-26B-A4B	Math	69.90	65.05	66.02	58.25	70.87	71.84	66.02	62.14	81.55	77.67	72.82	60.19
	Code	76.00	54.00	68.00	59.00	73.00	68.00	67.00	57.00	74.00	67.00	67.00	57.00
	DB	42.06	44.86	42.06	35.51	61.68	54.21	46.73	42.99	64.49	60.75	50.47	45.79
	Actions	40.38	31.73	36.54	28.85	84.62	79.81	65.38	81.73	82.69	75.96	83.65	75.96
2.5-7B-Instruct	Math	49.51	48.54	46.60	20.39	48.54	39.81	38.83	20.39	62.14	52.43	47.57	28.16
	Code	40.00	44.00	34.00	27.00	43.00	46.00	38.00	36.00	44.00	50.00	32.00	25.00
	DB	40.19	34.58	33.64	19.63	42.06	26.17	41.12	8.41	50.47	32.71	47.66	20.56
	Actions	47.12	50.00	49.04	38.46	65.38	65.38	34.62	51.92	65.38	64.42	47.12	31.73
3.1-8B	Math	33.98	29.13	22.33	17.48	50.49	38.83	20.39	33.98	54.37	48.54	33.98	29.13
	Code	25.00	21.00	16.00	10.00	23.00	23.00	9.00	18.00	27.00	24.00	22.00	16.00
	DB	22.43	24.30	16.82	19.63	41.12	33.64	28.97	18.69	47.66	40.19	38.32	25.23
	Actions	55.77	52.88	48.08	48.08	76.92	75.96	60.58	64.42	79.81	79.81	53.85	68.27
Mean		46.10	42.71	40.96	31.46	55.57	51.83	43.03	39.13	59.42	55.55	50.93	38.57
Overall Avg.		**40.31**				**47.39**				**51.12**			
Depth gain (pp)		$k=1$				$k=2$				$k=3$			$k=4$
vs. Base		+9.61				+11.74				+11.67			+10.64
vs. SFT		+2.91				+3.86				+2.49			+2.16

5 Related Work

Multi-Turn Dialogue. Multi-turn benchmarks assess instruction retention, context use, and answer refinement (Bai et al., 2024; Kwan et al., 2024; Deshpande et al., 2025). LiC reports degradation when tasks span turns (Laban et al., 2026). Intent Mismatch examines intent alignment through instruction reconstruction (Liu et al., 2026). Several benchmarks examine evolving requirements: SEQUOR and Evolving User Intent study requirement changes (Canaverde et al., 2026; Tack et al., 2026); SpecPath compares coding histories with equivalent final specifications, including cancelled requirements (Wu et al., 2026); Trip+ evaluates adaptation to changing travel requirements while preserving active commitments (Chen et al., 2026a). In contrast, *`Intent-Eval` compares clarification, rejection, and revision on the same tasks. Matched single-turn controls separate interaction effects from task difficulty, while longer dialogues test whether performance losses grow or persist.*

On-Policy Distillation and Self-Distillation. On-policy distillation supervises Student-generated prefixes with Teacher distributions (Agarwal et al., 2024). Self-Distilled Reasoner uses on-policy self-distillation (OPSD), giving reference solutions to a same-model Teacher (Zhao et al., 2026). On-policy context distillation (OPCD) gives the Teacher context unavailable to the Student (Ye et al., 2026). Found in Conversation (FiC) transfers single-turn competence, while MAIGO uses self-distillation with supervision from cleaned dialogue histories (Chen et al., 2026b; Zheng et al., 2026). *`Intent-OPSD` selects the Teacher's single-turn task according to the user's decision to retain or revise a task requirement and uses its supervision to train the Student on the full dialogue.*

6 Conclusion

`Intent-Eval` combines controlled comparisons of clarification, rejection, and revision with matched single-turn controls across four domains, revealing degradation that can deepen or persist in longer dialogues. Inactive-content errors under *Retained* and *Revised* suggest *mentioned-as-in-effect* confusion. `Intent-OPSD` provides active-intent supervision by aligning the Teacher's single-turn task with requirements remaining in effect, while the Student learns from the full dialogue. This improves mean accuracy over Base and matched SFT, without a Teacher at inference. Together, these findings highlight the need to distinguish what was *said* from what remains *in effect*.

AI USE STATEMENT

Generative AI tools were used for manuscript drafting and editing and to draft proposal, decision, and clarification templates. The authors reviewed these templates. Qwen3.6-27B generated simulated user turns for evaluation and training. Appendix B.2 describes the simulator checks, fallback mechanisms, and manual audit. The authors are responsible for the final text, claims, results, and research artifacts.

ETHICS STATEMENT

`Intent-Eval` uses existing benchmark tasks with controlled requirement changes and simulated user turns. The study evaluates model-generated responses and does not involve recruited human participants. Source benchmarks are cited throughout the paper.

REPRODUCIBILITY STATEMENT

Sections 2 and 3 describe the benchmark conditions and training objective. Section 4 presents the experimental setup and results. Appendix A provides details on task construction, reference validation, and domain-specific scoring. Appendix B gives the models, generation settings, seeds, user-simulator prompt, and evaluation procedure. Appendix C describes the interaction-depth protocols. Appendix D gives the training-source selection, training settings, and matched SFT setup. Appendix E.2 provides error diagnostics, and Appendix E.3 reports the interaction-depth cohorts and results. Reported tables and paired error counts were checked against the recorded model outputs. Code for `Intent-Eval` and `Intent-OPSD` is available in the anonymous repository linked in Section 4.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, et al. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7421–7454, 2024.

Beatriz Canaverde, Duarte M. Alves, José Pombal, Giuseppe Attanasio, and André F. T. Martins. SEQUOR: A multi-turn benchmark for realistic constraint following. *arXiv preprint arXiv:2605.06353*, 2026. doi: 10.48550/arXiv.2605.06353. URL https://arxiv.org/abs/2605.06353.

Junle Chen, Wei Chen, Yehong Xu, Zhengjun Huang, Yuqian Wu, Zhoujin Tian, Kai Wang, Lei Wang, and Xiaofang Zhou. Trip+: Benchmarking agents in personalized interactive travel planning. *arXiv preprint arXiv:2606.21169*, 2026a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Tianlang Chen, Shirley Wu, and Jure Leskovec. Found in conversation: Llms teach themselves to close the multi-turn gap. *arXiv preprint arXiv:2605.24432*, 2026b.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Wai-Chung Kwan, Xingshan Zeng, Yuxin Jiang, Yufei Wang, Liangyou Li, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. Mt-eval: A multi-turn capabilities evaluation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 20153–20177, 2024.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VKGTGGcwl6.

Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

Geng Liu, Fei Zhu, Rong Feng, Changyi Ma, Shiqi Wang, and Gaofeng Meng. Intent mismatch causes llms to get lost in multi-turn conversation. *arXiv preprint arXiv:2602.07338*, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Jihoon Tack, Philippe Laban, and Jennifer Neville. Llms get lost in evolving user intent. *arXiv preprint arXiv:2607.20734*, 2026.

Yangfan Wu, Haozhe Wang, Huanyu Yang, Jianmin Ji, and Fangzhen Lin. Specpath: Testing coding agents across contract-equivalent specification histories. *arXiv preprint arXiv:2608.09799*, 2026.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3911–3921, 2018.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Haoyu Zheng, Yun Zhu, Shu Yuan, Shangming Chen, Qing Wang, Wenqiao Zhang, Jun Xiao, and Yueting Zhuang. Maigo: Mitigating lost-in-conversation with history-cleaned on-policy self-distillation. *arXiv preprint arXiv:2605.27186*, 2026.

SUPPLEMENTARY MATERIAL

You Changed Your Mind, The Model Didn't: Demystifying Intent in Multi-Turn Dialogue

TABLE OF CONTENTS

A Construction Details, Scoring, and Examples **13**

A.1 Construction details . . . 13

A.2 A complete worked example . . . 15

B Evaluation Settings and User Simulation **17**

B.1 Models and generation settings . . . 17

B.2 User simulation . . . 17

B.3 User simulator reliability . . . 18

C Interaction Extension Protocols **19**

C.1 Accumulation: repeated clarification or rejection . . . 19

C.2 Persistence: clarification after a decision . . . 19

D `Intent-OPSD` Training and Evaluation **19**

D.1 Training data and conversation construction . . . 19

D.2 Training objectives and generation budgets . . . 20

E Supplementary Results and Error Analysis **21**

E.1 Overall performance (RQ1) . . . 21

E.2 Error diagnostics and examples (RQ2) . . . 21

E.3 Interaction depth in six additional models (RQ3) . . . 22

E.4 Intent-OPSD across interaction depths (RQ4) . . . 23

F Limitations and Discussion **24**

A Construction Details, Scoring, and Examples

Section A.1 supplements the construction procedure in Section 2.2 (Steps I–IV) with domain-specific editing rules, candidate checks and selection, reference construction, and condition rendering. It also describes scoring for each domain. Section A.2 follows one task through the complete process, including the messages and reference answers used in evaluation.

A.1 Construction details

Table 3 summarizes the edits and reference updates. We apply the construction constraints in Section 2.2. We discard candidates with invalid references or whose original and revised task states cannot be distinguished by the applicable verifier or output-contract checks.

Table 3: Domain-specific revisions and reference updates.

Domain	Revised requirement	Reference update
Actions	Tool argument in specified calls	Replace the affected argument values
Code	Entry-point function or method name	Rename the callable in the contract and available reference code
Database	Query condition or output field	Update the corresponding SQL clause
Math	Numerical premise	Recompute with the revised value

Actions. The 104 Actions revisions change one tool argument in a call or a specified group of calls. Each proposal identifies the affected calls. We check the edited argument's type and allowed values against the tool schema. The updated calls must also satisfy any total specified in the task. We exclude changes to shared settings, such as units, that would alter the meaning of other argument values. When several targets are valid, we prefer one introduced before the final source passage so that task information remains to be revealed after the event.

After updating the affected reference arguments, we check both task states. Each reference must pass its own verifier and fail the verifier for the other state. To score a response, we parse it into function calls and compare their structure and argument values with those allowed by the active task.

Example. In `sharded-BFCL/parallel_143`, the user asks for the total area of three triangular gardens. The base and height pairs are $(10, 5)$, $(15, 7)$, and $(20, 10)$ meters. The proposal and the affected reference call are shown below:

Actions: proposal and affected reference call

Proposal: For the `calc_area_triangle` call with base 10, use 6 for the `height` argument. Would this be suitable?

Original: `calc_area_triangle(base=10, height=5)`
Revised: `calc_area_triangle(base=10, height=6)`

The selected reference call uses `height=6` only if the proposal is accepted. Otherwise, it keeps `height=5`. The other two calls, for pairs $(15, 7)$ and $(20, 10)$, stay unchanged.

Code. The 100 Code revisions change the required entry-point function or method name recorded in the task metadata. The original name appears in the first source message, and a later proposal specifies the new name.

We update the name in the task contract and any available reference code, leaving the computation unchanged. Both task states use the same verifier and execution tests, with different naming requirements. When both reference programs are available, each must pass under its corresponding task state and fail under the other. Otherwise, we check that the specified name change has been applied.

To evaluate generated programs, we check their entry-point names and run the execution tests. In both single- and multi-turn conditions, a program must use the new name if the proposal is accepted and the original name otherwise. Defining both names is not allowed.

Example. In `sharded-HumanEval/5`, the task inserts a delimiter between adjacent numbers in a list. The box shows the proposal to rename the function, followed by a test call to each reference program:

Code: proposal, reference calls, and shared output

Proposal: Use `intersperse_revised` as the required function name. Would you recommend this option?

Original: `intersperse([5, 6, 3, 2], 8)`
Revised: `intersperse_revised([5, 6, 3, 2], 8)`
Result in both cases: `[5, 8, 6, 8, 3, 8, 2]`

The required function name changes only if the proposal is accepted. Otherwise, it remains `intersperse`. The function body and expected outputs remain unchanged.

Database. We select a query requirement in the original single-turn task. A candidate proposes a new value for that requirement and updates the corresponding SQL condition. We also consider changes to ordering, set membership, comparisons, aggregation, and output fields, in that order. If the initial rules yield no valid candidate, we keep the first valid candidate from these additional edits. We run the original and revised reference queries on the task's database. If they return the same result, we discard the revision. The 107 events comprise 88 SQL-condition changes and 19 output-field changes.

Generated queries are scored by comparing their execution results with the active reference on that database instance.

Example. In `sharded-spider-val-265-medium`, the user asks for the cities of origin of employees under age 30, keeping only cities represented by more than one qualifying employee. The proposal and the two reference queries are shown below:

Database: proposal and reference queries

Proposal:
the employees in question should be under the age of 32. How would this revised requirement work?

Original:
SELECT city FROM employee WHERE age $<$ 30
GROUP BY city HAVING count($*$) $>$ 1

Revised:
SELECT city FROM employee WHERE age $<$ 32
GROUP BY city HAVING count($*$) $>$ 1

Only the age cutoff changes, from 30 to 32. **On the task's database, the reference result changes to Bath and Bristol only if the proposal is accepted. Otherwise, it remains Bath.**

Math. For each of the 103 Math tasks, we replace one numerical premise in the original single-turn task. For rule-based edits, the selected passage must contain a single numerical target whose value does not appear in other passages. We exclude numbers used as labels and replacements that conflict with other quantities.

We recompute the revised answer and check that both reference answers are valid and different. To score a response, we extract its final number, normalize its format, and compare it with the reference for the active task.

Example. In `sharded-GSM8K/799`, bread costs $2 per loaf and bagels cost $1 each. The task asks how much more three loaves cost than two bagels.

Math: proposal and reference answers

Proposal: each bagel costs $2. Would this be a suitable update?
Original: $3 \times \$2 - 2 \times \$1 = \$4$
Revised: $3 \times \$2 - 2 \times \$2 = \$2$

The reference answer changes to $2 only if the proposal is accepted. Otherwise, it remains $4.

Selecting a revision. When several rule-based candidates remain for a task, we first choose the requirement to change. We apply the domain-specific priorities above, then favor longer source passages and use earlier positions to break ties. For that requirement, a fixed rule based on the source task ID selects one valid replacement. Selection is independent of model outputs.

Human review and condition rendering. We review each selected revision alongside its source task to check that the change is clear and consistent with the other requirements. We use LLMs to draft templates with varied wording for proposals, decisions, and clarification questions. After human review, we use these templates to render approved revisions into dialogue messages. We save the target passage, resulting messages, and revised reference in one record.

Retained and Revised share the proposal, paired with rejection or acceptance, respectively. Neutral uses two template-based clarification questions about the same requirement, and Original has no inserted event. Where multiple template variants are available, a stable rule based on the record identifier selects the wording.

We insert the event after the message that introduces the target requirement, then continue with any remaining task information. Each condition uses the reference specified in Section 2.1.

Neutral questions refer to the requirement being revised. Automated checks confirm that they reveal neither the original nor the replacement value, that Retained and Revised share the same proposal, and that decision messages do not repeat the proposed value. All 414 selected revisions pass these checks.

A.2 A COMPLETE WORKED EXAMPLE

We now walk through one evaluated task. The onion-purchase task in `sharded-GSM8K/248`, shown in Figure 1, provides the source prompt and four shards below. All conditions share the same **Math** system message.

Math System Message

As an expert problem solver solve step by step the following mathematical questions.

Original Single-Turn Prompt (S_{Orig})

Q: A chef bought **4 bags** of onions. Each bag weighs 50 pounds. A pound of onions cost $1.50. How much did the chef spend?
A:

Source Shards

S1: calculate how much the chef spent on onions
S2: the chef bought **4 bags** of onions
S3: each of those bags weighs 50 pounds
S4: and each pound of onions costs $1.50

1. Selecting the target and updating the reference. The selected target is the **bag count** in S2, which changes from **four** to **two**. The weight per bag, price per pound, and spending question remain unchanged. Recomputing the cost gives

$$y^{(0)} = \mathbf{4} \times 50 \times \$1.50 = \mathbf{\$300}, \qquad y^{(1)} = \mathbf{2} \times 50 \times \$1.50 = \mathbf{\$150}.$$

2. Placing the event in the dialogue. Each labeled item is sent as a separate user message. The labels are included here only to guide the reader.

Proposal and Decisions for This Example

Propose (shared): the chef bought **2 bags** of onions. Does this alternative make sense?
Reject (Retained): Ignore that revision.
Accept (Revised): Let it change as proposed.

The reference answer changes to $150 only if the proposal is accepted. Otherwise, it remains $300.

Canonical Neutral Messages

Neutral 1: How does the **number of onion bags** the chef bought enter the calculation?
Neutral 2: Why is the **number of onion bags** the chef bought important for determining the requested result?

The event follows S2, once the bag count has been introduced. S3 and S4 then supply the weight and price needed for the calculation, as shown in Figure 1. The user's decision determines which bag count is active (Table 4). The post-decision Neutral suffix reuses the two clarification questions above.

Table 4: Condition-specific targets for the worked Math example.

Condition	Inserted event	Active bag count	Reference answer
Original	None	**4**	**$300**
Neutral	Clarification questions	**4**	**$300**
Retained	Propose **2 bags**, then reject	**4**	**$300**
Revised	Propose **2 bags**, then accept	**2**	**$150**

In evaluation, each conversation is generated independently using the event messages reproduced above.

3. Single-turn controls. Each box contains one complete user message, including the connectors that introduce the proposal and decision.

Direct Revised Single-Turn Prompt (S_{Dir})

Q: A chef bought **2 bags** of onions. Each bag weighs 50 pounds. A pound of onions cost $1.50. How much did the chef spend?
A:

Revised Single-Turn Prompt (S_{Rev})

Q: A chef bought **4 bags** of onions. Each bag weighs 50 pounds. A pound of onions cost $1.50. How much did the chef spend? Before you go, I have one quick last minute thought: The chef bought **2 bags** of onions. **Okay, let's do that.**
A:

Retained Single-Turn Prompt (S_{Ret})

Q: A chef bought **4 bags** of onions. Each bag weighs 50 pounds. A pound of onions cost $1.50. How much did the chef spend? Before you go, I have one quick last minute thought: The chef bought **2 bags** of onions. **Wait, that does not seem quite right. Ignore that revision.**
A:

B Evaluation Settings and User Simulation

B.1 Models and generation settings

Model versions. For Llama-3.1-8B, we use the Unsloth distribution of Meta-Llama-3.1-8B-Instruct. The Gemma model is Gemma-4-26B-A4B-it.

Table 5: Generation settings for local models and the user simulator. Output limits are in tokens per response.

Setting	Local single-turn	Local multi-turn
Output limit	1,000	4,096
Temperature	0	0
Top-p	1	1
Seed	0	Derived from seed 0
User simulator (all multi-turn runs)		
Model	Qwen3.6-27B	
Temperature	0.3	
Output limit	192	

API models use default sampling settings. Their single-turn and multi-turn output limits match those in Table 5, including reasoning tokens.

Scoring and aggregation. The domain verifier in Appendix A.1 scores the final eligible response after source disclosure and the event path are complete. Benchmark accuracy pools sources within each model and then averages models with equal weight.

B.2 User simulation

Following LiC (Laban et al., 2026), a *shard* is a short text fragment containing part of a task, such as its goal or one requirement. To *reveal* a shard is to convey its information to the assistant in a user message. The *user simulator* is the LLM used to generate user messages, rephrasing shards to fit the ongoing dialogue. The four multi-turn conditions are matched by source task. The evaluation protocol directly inserts the first shard. The user simulator then generates messages based on the evolving dialogue history. After the relevant requirement has been introduced, the protocol inserts the fixed *event messages* specified for that condition, including clarifications, proposed changes, and acceptance or rejection messages.

An *eligible shard* has not yet been revealed and is allowed by the protocol on the current turn. After the inserted event in *Neutral*, *Retained*, or *Revised*, the simulator must resume task disclosure by revealing an eligible shard. We call this a *mandatory-disclosure turn*. The script checks the JSON reply's format, selected shard ID, and text, rejecting empty replies and bare acknowledgments. If a check fails, it uses a *fallback*: the stored text of the first eligible shard is sent unchanged. When consecutive replies reveal no new shard, a recovery step likewise requires the simulator to disclose an eligible shard.

For each simulator call, the evaluation script fills the following inputs:

- `[[CONVERSATION_SO_FAR]]`: the user and assistant messages so far, excluding the assistant's system instructions.
- `[[SHARDS_REVEALED]]`: shards already supplied in user messages, listed with their IDs and text.
- `[[SHARDS_NOT_REVEALED]]`: shards not yet supplied and available on this turn. Shards held back for later turns are excluded.

The JSON reply has two fields: `response` contains the next user message, and `shard_id` identifies the shard it reveals (-1 when none is revealed). Only `response` is sent to the evaluated model.

User Simulator Prompt (LiC-Derived)

You are simulating a user of an interactive LLM system (like ChatGPT).
The user is inherently lazy, and answers in short form, providing only minimal information to the system. You should not be proactive.

Here's the conversation so far:
[[CONVERSATION_SO_FAR]]

Here are the shards that have already been revealed:
[[SHARDS_REVEALED]]

Here are all the shards that have not been revealed yet:
[[SHARDS_NOT_REVEALED]]

You must generate a response to the conversation so far. Here are the rules:
– [Providing a Shard] You can reveal the content of a shard to the system in your response if it will help the system move closer to answering the problem. You should select the shard to reveal that is most "basic" and is the current most relevant shard.
– [One Shard at a Time] You should only reveal at most one shard at a time.
– [Reveal Entire Shard] If you reveal a shard, you must make sure to include ∗all the information in the shard∗. For example, if the shard is "your symptoms are that you have a headache in the mornings", your response can't just be ''yeah I have headaches'', you must say ''yup mostly headaches in the mornings''.
– [Irrelevant Clarifications] If the system asks you a question irrelevant to the shards, asks you a generic question (''Can you give me a hint?''), you should respond with an answer that does not provide a shard. (''I don't know'', ''Is that really important?'', etc.) You should not reveal any information beyond what is available in the shards.
– [No Repeated Shards] You should not reveal the same shard more than once. Carefully review the shards revealed already, and only reveal a shard if its 'shard_id' is not on the list.
– [Rephrase Shards] If you reveal a shard, you should rephrase it in a conversational way. Do not copy the shard verbatim.
– [Do Not Ask Questions] Your response should always be declarative sentences, and not questions.
– [Brevity of Response] You should favor being succint. Your answer can also have typos, improper grammar, capitalization, etc. You are simulating a real person talking to an AI, who is in a hurry.
– [Task Changes] Task changes and accept/reject decisions are supplied separately. Do not introduce or reverse them, or restore source values superseded by an accepted change.
– Use only facts from the source shards. Ignore anything the Assistant calculated, guessed, or answered; do not repeat or confirm it as a user fact.
– [Format] Your response should be formatted as a JSON object with the following keys:
  – 'response': The response to the conversation so far.
  – 'shard_id': The shard you are revealing to the system. The shard_id can be an integer, or −1 if you did not reveal any shards.
For example:
{"response": "I don't know", "shard_id": −1}
or:
{"response": "yeah I want it to [...]", "shard_id": 1}

On turns that must reveal new task information, the evaluation script adds the following instruction:

Mandatory-Disclosure Instruction

Current turn requirement: reveal exactly one unrevealed shard. Return its shard_id, not −1; do not give only an acknowledgement.

B.3 User simulator reliability

We evaluated the LLM-based user simulator, including its fallback and recovery mechanisms. A manual inspection covered **2,345 simulated user turns** across 500 recorded conversations, with 125 conversations from each of **Actions**, **Code**, **Math**, and **Database**. We adapted the user-turn inspection dimensions of Laban et al. (2026) to assess messages actually delivered to the assistant.

Task-relevant information delivery checks whether a shard-bearing message conveys its task-relevant content in the context available at that turn. Paraphrasing, omission of task-irrelevant background, and faithful combinations of shards are accepted. This criterion was met in **2,284**

of 2,317 shard-bearing messages (98.6%). **Contextualization** checks whether a message appropriately addresses the preceding assistant response. All **2,345 inspected messages (100%)** were judged appropriately contextualized.

C Interaction Extension Protocols

Section 2.3 defines the single-turn controls; Appendices A.2 and E.1 provide a complete prompt example and numerical comparisons, respectively.

C.1 Accumulation: repeated clarification or rejection

We write the ℓth clarification block as $N^{(\ell)} = [n_1^{(\ell)}, n_2^{(\ell)}]$, which contains two Neutral messages about the same selected requirement, without introducing a replacement value. The corresponding Retained block is $P^{(\ell)} = [\textsc{Propose}(r'_\ell), \textsc{Reject}]$. Each block proposes a different replacement for that requirement and then rejects it. Thus, the requirement being discussed stays fixed across blocks, and the original requirement remains active after every rejection.

The operator $\diamond$ joins the blocks in order. Each event message is sent as a separate user turn, followed by an assistant reply. The operator $\oplus_j$ inserts the sequence after the assistant's reply to u_j, the user message that first introduces the selected requirement, and before the remaining task information is disclosed. With k blocks, the two paths are

$$\begin{aligned} M_{\text{Neu}}^{(k)} &= M_{\text{Orig}} \oplus_j (N^{(1)} \diamond \cdots \diamond N^{(k)}), \\ M_{\text{Ret}}^{(k)} &= M_{\text{Orig}} \oplus_j (P^{(1)} \diamond \cdots \diamond P^{(k)}), \\ M_{\text{Neu}}^{(0)} &= M_{\text{Ret}}^{(0)} = M_{\text{Orig}}. \end{aligned} \tag{6}$$

Both paths add $2k$ event user messages.

C.2 Persistence: clarification after a decision

For $v \in \{\text{Ret}, \text{Rev}\}$, let $D_v = [\textsc{Propose}(r'), \textsc{Decision}_v]$ denote the canonical proposal followed by rejection or acceptance, sent as two separate user messages and held fixed across depths. The k Neutral blocks begin after the assistant's reply to the decision message, before any remaining task information is disclosed:

$$\begin{aligned} M_{v\to\text{Neu}}^{(k)} &= M_{\text{Orig}} \oplus_j (D_v \diamond N^{(1)} \diamond \cdots \diamond N^{(k)}), \\ M_{v\to\text{Neu}}^{(0)} &= M_v. \end{aligned} \tag{7}$$

The same clarification questions are used after either decision. They usually concern the selected requirement without proposing new values. These questions do not change the active task: Retained keeps the original requirement active, while Revised keeps the accepted revision active at every depth.

D Intent-OPSD Training and Evaluation

Active-task reference construction. FiC (Chen et al., 2026b) constructs its Teacher input from concatenated shards, while MAIGO (Zheng et al., 2026) removes assistant replies and adds a complete-task reference at the answer turn. Applying these constructions here requires specifying which requirements remain active, since user messages can also contain rejected or superseded content. `Intent-OPSD` explicitly aligns supervision with the user's decision: rejection selects the original Teacher task, while acceptance selects the revised task. The Student receives the full dialogue and is trained to follow the resulting active requirements.

D.1 Training data and conversation construction

Training, validation, and evaluation use disjoint source IDs. We provide 50 validation sources per domain. The **Actions**, **Database**, and **Math** training pools derive from BFCL (Patil et al., 2025),

Spider (Yu et al., 2018), and GSM8K (Cobbe et al., 2021), respectively. The **Code** pool contains 193 MBPP (Austin et al., 2021) tasks, 56 HumanEval (Chen et al., 2021) tasks, and 96 easy and 155 medium LiveCodeBench (Jain et al., 2025) tasks.

We initially assemble 500 training candidates per domain and apply domain-specific data checks. For **Database**, we check that the original and revised queries match their respective task requirements and produce different results on the task database. These checks leave 381 **Database** candidates. The resulting pools then undergo the Teacher admission check in Section 3.1, using a frozen copy of the Student's base model as the Teacher. Table 6 reports the resulting training counts for each model and domain.

Table 6: Teacher-admitted training sources by model and domain. Each source contributes all four conditions.

Model	Actions	Code	Database	Math
Qwen3-8B	435	316	201	451
Gemma-4-26B-A4B	447	367	307	428
Qwen2.5-7B-Instruct	323	322	214	454
Llama-3.1-8B	407	266	212	455

Training conversations follow the sharded conversation setup of LiC (Laban et al., 2026). Each complete task is divided into information shards, and a user simulator reveals at most one shard per turn while interacting with the current Student. We add the benchmark's condition-specific clarification, proposal, and decision messages to these conversations. The resulting histories are generated online as the Student changes during training.

D.2 Training objectives and generation budgets

Both `Intent-OPSD` and matched SFT use LoRA (Hu et al., 2021), with shared training hyperparameters reported in Table 7.

`Intent-OPSD`. Training uses the final-answer distillation objective in Equation 4. Correct and incorrect Student answers both contribute to the loss, which is averaged over conditions and sources.

Matched SFT. SFT reuses the selected sources and dialogue histories H_v recorded during `Intent-OPSD` training. For each condition v, its target y_v^{T} is the verified Teacher answer recorded during admission for the corresponding complete task c_v (Equation 3). For each source, SFT minimizes

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\frac{1}{4}\sum_{v}\frac{1}{|y_v^{\mathrm{T}}|}\sum_{t=1}^{|y_v^{\mathrm{T}}|}\log \pi_\theta\big(y_{v,t}^{\mathrm{T}} \mid H_v, y_{v,<t}^{\mathrm{T}}\big)\,. \tag{8}$$

Here, v ranges over the four conditions. The loss is averaged over sources, and history tokens provide context without receiving supervision. Both methods start from the same base model and use the same training sources, condition weights, and optimization schedule. Both methods receive supervision aligned with the user's final decision. We compare on-policy distribution matching with hard-target learning under shared active-task supervision.

Training budgets. For **Code**, **Database**, and **Math**, the maximum training sequence length is 32,768 tokens, with up to 4,096 tokens for both intermediate replies and final answers. **Actions** uses 8,192 tokens per training sequence, 128 per intermediate reply, and 512 per final answer. Its output format requires function calls without explanatory text, motivating a more compact training budget. Evaluation uses the common multi-turn generation settings in Appendix B.

Table 7: Training hyperparameters shared by `Intent-OPSD` and matched SFT within each model–domain pair. Each source contributes all four conditions.

Hyperparameter	Setting
Optimizer	AdamW
Learning rate	1×10^{-5}
Weight decay	0
Learning-rate schedule	Constant, without warmup
Training epochs	1
Sources per optimizer update	4 (Actions, Database); 8 (Code, Math)
LoRA rank / alpha / dropout	16 / 32 / 0
Training seed	42

E Supplementary Results and Error Analysis

We follow the research questions in order: additional models and single-turn controls for RQ1, error patterns for RQ2, interaction-depth analysis for RQ3, and distillation results across depths for RQ4.

E.1 Overall performance (RQ1)

Additional model results. Table 8 provides supplementary results for Qwen3-4B-Instruct-2507 and Qwen2.5-7B-Instruct under the same conditions as Table 1. These two models are not included in the eight-model Overall aggregate in Table 1. In every domain, both models score lower on Multi Original than on Single Original, and lower again on Multi Retained.

Table 8: Full results for Qwen3-4B-Instruct-2507 and Qwen2.5-7B-Instruct under the same conditions as Table 1.

Model	Domain	Single				Multi			
		Original	Direct Rev.	Revised	Retained	Original	Neutral	Revised	Retained
3-4B	Math	86.41	87.38	80.58	76.70	54.37	53.40	50.49	27.18
	Code	78.00	79.00	42.00	77.00	28.00	30.00	28.00	24.00
	Database	90.65	89.72	74.77	69.16	37.38	28.04	37.38	22.43
	Actions	92.31	91.35	76.92	84.62	48.08	48.08	43.27	34.62
2.5-7B	Math	89.32	87.38	73.79	63.11	49.51	48.54	46.60	20.39
	Code	68.00	69.00	62.00	43.00	40.00	44.00	34.00	27.00
	Database	81.31	79.44	74.77	40.19	40.19	34.58	33.64	19.63
	Actions	97.12	94.23	72.12	75.00	47.12	50.00	49.04	38.46

E.2 Error diagnostics and examples (RQ2)

We analyze the multi-turn outputs in Tables 1 and 8, selecting cases scored correctly in Original but incorrectly in Retained or Revised. *Inactive content* refers to the rejected proposal in Retained or the original requirement replaced in Revised. We apply domain-specific automatic checks to the complete final response: argument values in **Actions**, entry-point identifiers in **Code**, SQL literals, operators, or predicates in **Database**, and premise or reference-answer values in **Math**. Explanations and code comments are included. We report the fraction of selected error responses that contain inactive content.

Multi-turn results. Across all ten models, inactive content appears in 446 of 748 new Retained errors (59.63%) and 201 of 649 new Revised errors (30.97%). Figure 6 provides the model-level breakdown.

Examples. The examples below show rejected or superseded values used in calculations that produce incorrect final answers.

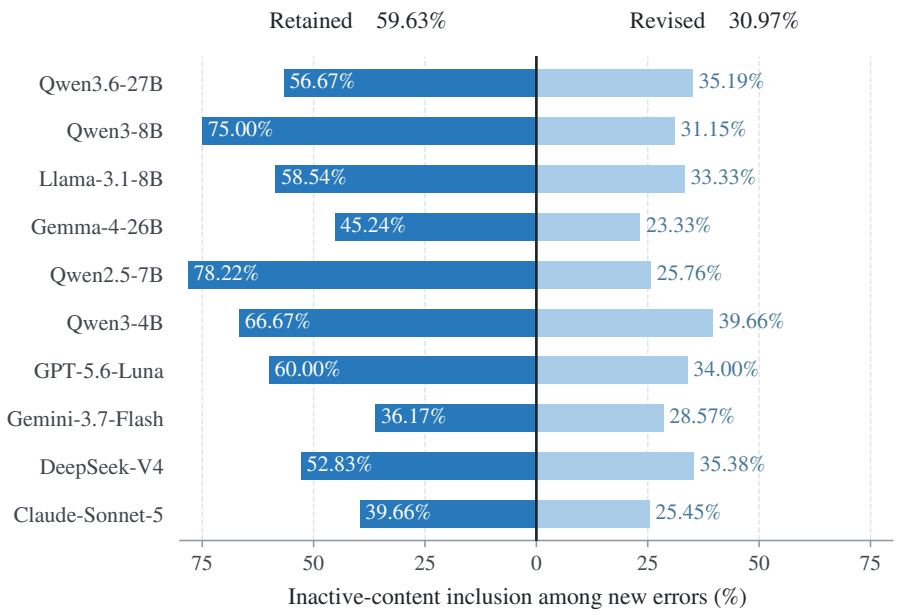


Figure 6: Inactive content appears more often in Retained errors for all ten models. Bars show multi-turn content-inclusion rates among Original-correct outputs that become incorrect in the named condition. Headers pool all ten models.

Example 1: using a rejected value

Model: GPT-5.6-Luna, multi-turn Retained.
Source: `sharded-GSM8K/543`.
Task (summary): Jairus earns **$0.80 per task** and Jenny earns $0.50. Each completes 20 tasks. How much more does Jairus earn?
Proposed change (summary): Jairus earns **$0.90 per task**.
User decision: "That revision does not work for me."
Active answer: **$6**.
Response (summary): Uses **$0.90 per task** and returns **$8**.

Example 2: keeping the original value after acceptance

Model: Claude-Sonnet-5, multi-turn Revised.
Source: `sharded-GSM8K/698`.
Task (summary): Greta receives a $2,400 paycheck, puts 50% into retirement, and spends **20% on her car payment**. How much remains?
Proposed change (summary): Spend **21% on the car payment**.
User decision: "Authorize that adjustment."
Active answer: **$696**.
Response (summary): Lists the car payment as **20% ($480)** and reports **$720** remaining.

E.3 Interaction depth in six additional models (RQ3)

Figure 7 complements Section 4.3 with six additional models evaluated on the same 372-source accumulation and 414-source persistence cohorts.

These results support the main-text trends. Averaged across the six models and four interaction paths, the first added block produces the largest single-step accuracy drop. Repeated Retained produces larger average losses than Neutral, with pronounced declines in Math. After either de-

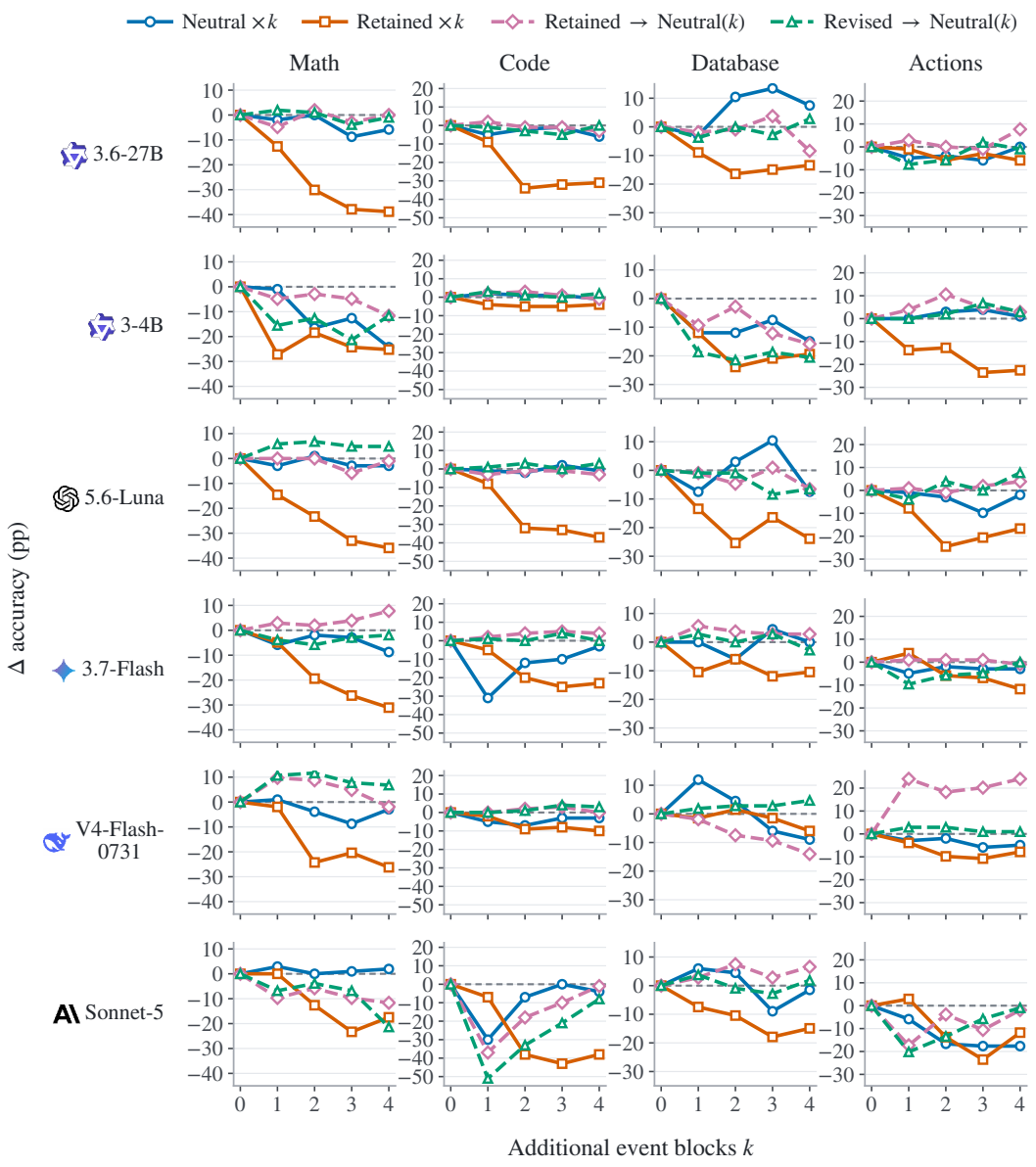


Figure 7: Interaction depth in six additional models. Accuracy changes are relative to each curve's baseline at $k = 0$.

cision, additional clarification also leaves mean accuracy below the corresponding baseline at the final depth, with effects varying across models and domains.

E.4 INTENT-OPSD ACROSS INTERACTION DEPTHS (RQ4)

We evaluate the four RQ4 models on 372 accumulation and 414 persistence sources per model, pooling counts within each path and depth (Table 9). Accumulation starts from Original; persistence starts from the respective decision baseline. Code inputs specify the original entry-point name before revision proposals.

`Intent-OPSD` exceeds Base at every tested depth in all four pooled paths. Pooling the four paths yields gains of 9.61–11.74 pp over Base and 2.16–3.86 pp over SFT at $k = 1$–4.

Table 9: Base, matched SFT, and `Intent-OPSD` accuracy (%) across interaction depths. Each path pools 1,488 evaluations for accumulation or 1,656 for persistence; Mean pools all four paths at each depth. Colors follow Table 2, comparing each method with Base at the same depth.

Interaction path	Method	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
Accumulation: Neutral	Base	47.58	43.95	41.53	40.12	39.45
	SFT	55.98	52.69	52.49	51.88	52.28
	`Intent-OPSD`	60.62	56.79	57.86	56.65	56.99
Accumulation: Retained	Base	47.58	32.19	27.08	25.87	26.81
	SFT	55.98	40.32	38.17	37.43	38.58
	`Intent-OPSD`	60.62	38.84	37.43	35.89	35.22
Persistence: Retained	Base	31.40	30.07	28.56	27.78	28.68
	SFT	39.07	37.26	35.21	35.51	34.54
	`Intent-OPSD`	38.59	37.86	36.17	35.08	34.78
Persistence: Revised	Base	40.88	38.04	37.26	35.27	37.38
	SFT	43.06	41.12	40.70	41.43	41.61
	`Intent-OPSD`	50.97	49.21	50.24	48.43	48.37
Mean	Base	41.56	35.96	33.57	32.22	33.08
	SFT	48.12	42.65	41.44	41.40	41.56
	`Intent-OPSD`	52.27	45.56	45.31	43.89	43.72

F Limitations and Discussion

Interaction coverage. `Intent-Eval` uses controlled clarification, rejection, and revision events to compare responses to the same source tasks under different interaction conditions. This design makes the active requirements and reference answers explicit and verifiable. The current benchmark focuses on decisions about one selected requirement. Extending coverage to multiple interacting requirements and more varied user behavior remains future work.

Method development. `Intent-OPSD` provides a simple training approach to help LLMs identify and follow active intent in multi-turn dialogue. It improves overall accuracy over Base and matched SFT in our experiments. However, mean *Retained* accuracy is similar to SFT. Handling rejected changes remains challenging and is an important direction for further method development.

Supervision from natural conversations. Our training uses complete active-task specifications supplied by the benchmark construction. These specifications support the Teacher during training, while the trained Student uses only the dialogue at inference. Applying the approach to natural conversations requires reconstructing active requirements from user messages and decisions. Future work could construct these specifications by extracting and updating requirements from dialogue, with consistency checks and human review for ambiguous cases.