DRIFT DOMINATES CONTRADICTION IN MULTI-TURN CONSTRAINT REASONING

Sebastien Kawada
Kaons
Los Angeles, United States
sebastien@kaons.org

ABSTRACT

How do multi-turn reasoning systems fail? The expected answer is logical contradiction, in which the system's maintained state becomes unsatisfiable. We show that the dominant mode is instead *satisfiable drift*, where the internal state stays consistent while the returned answer silently violates prior commitments. We build a solver-instrumented benchmark covering three constraint domains and 816 test problems, and evaluate four methods across four open-weight models (8B–120B parameters). MUS-Repair, which feeds minimal unsatisfiable subsets back to the generator, is strongest in every setting ($+1.8$ to $+15.0$ pp over the best non-MUS baseline). But the central finding is what repair leaves behind. After structured feedback, models rarely contradict themselves; they forget. Residual errors are 68–95% satisfiable drift across all settings, while contradiction drops to near zero. Reliable multi-turn systems must separately validate that the returned answer respects the maintained state.

1 INTRODUCTION

When an interactive assistant manages evolving structured state, it must honor every commitment it has already accepted while folding in new constraints. A scheduling tool that confirms "Bob is not on Tuesday" should never subsequently place Bob on Tuesday, yet current language models do exactly this with troubling regularity. What makes the failure especially dangerous is its subtlety. The system's internal state remains logically consistent, no solver alarm fires, and the returned answer looks correct to every automated check that inspects only state consistency. We call this pattern *satisfiable drift*, and show that it accounts for the vast majority of residual errors even after structured repair feedback (Figure 1, Table 1).

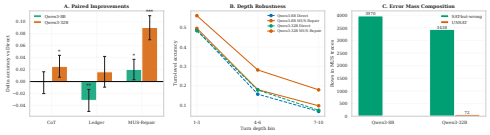


Figure 1: Contradiction vs. satisfiable drift. The constraint ledger remains satisfiable, so the solver raises no alarm, but the returned assignment silently violates an active constraint.

	Best baseline		MUS-REPAIR		
Model	Acc.	Method	Acc.	Drift %	Δ (pp)
Qwen3-8B	28.2	Direct	30.0	94.5	+1.8
Qwen3-32B	31.4	CoT	38.2	89.1	+6.8
gpt-oss-20b	53.7	Ledger	68.7	68.3	+15.0
gpt-oss-120b	54.0	CoT	62.7	78.4	+8.7

Table 1: Summary of main results. MUS-REPAIR outperforms the strongest non-MUS baseline in every setting. Drift % shows the share of residual errors from satisfiable drift rather than contradiction.

Existing evaluations collapse two fundamentally different failure modes into a single accuracy number (Wei et al., 2022; Yao et al., 2024; Madaan et al., 2023). *Contradiction*, where the maintained state becomes unsatisfiable, is a state-level defect that formal methods can detect. *Satisfiable drift*, where the state is consistent but the assignment violates it, requires a second verification layer most systems lack. This paper separates the two with a solver-instrumented benchmark that checks both ledger satisfiability and assignment validity at every turn across 816 problems and four open-weight models (Table 1).

Findings. ❶ MUS-REPAIR is the strongest method in every setting, producing gains of $+1.8$ to $+15.0$ pp over the best non-MUS baseline, all of which survive paired tests after false-discovery correction. ❷ These gains do not eliminate the dominant failure mode. After structured feedback, 68–95% of remaining failures involve a consistent ledger with a violating assignment, while contradiction drops to near zero. Models stop contradicting themselves but keep forgetting prior commitments. ❸ The degradation with conversational depth is structural rather than a capacity bottleneck. Even gpt-oss-120b drops from 93% at turn one to 40% at turn ten; higher capability lifts the entire curve but does not flatten it.

Contributions. ❶ A **solver-instrumented multi-turn benchmark** covering three constraint domains (logic grid, scheduling, seating) with Z3-verified turn-level decomposition of contradiction and drift. ❷ A **trigger-conditioned repair interface** that routes unsatisfiable states through MUS localization and satisfiable assignment failures through policy diagnostics within a single retry loop. ❸ The **first empirical demonstration** that satisfiable drift dominates residual errors across all tested settings, arguing that contradiction and drift should be reported as separate evaluation metrics.

2 RELATED WORK

Evaluation of multi-step reasoning. Prompting strategies, search over intermediate traces, and tool-augmented agent architectures have produced substantial accuracy gains on reasoning benchmarks (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2023; Yao et al., 2023a; Gao et al., 2023; Chen et al., 2023; Yao et al., 2023b; Hu et al., 2024; Han et al., 2025). These advances primarily target single-turn performance or final-answer quality, and they have achieved impressive results in that scope. However, most evaluations do not instrument turn-level state validity under accumulated constraints. The COLLIE benchmark (Yao et al., 2024) evaluates LLMs on constraint satisfaction, but it operates in a single-turn setting without multi-turn state tracking or failure-channel decomposition. Long-context reliability studies document degradation with depth (Press et al., 2022; Liu et al., 2024), yet they do not separate state inconsistency from assignment inconsistency conditional on a satisfiable state. Our benchmark is designed to fill this gap. Each turn is solver-verified for both ledger satisfiability and assignment validity.

Verifier-guided repair and self-correction. Iterative self-correction with verifier feedback produces strong aggregate improvements in mathematics and code (Cobbe et al., 2021; Lightman et al., 2023; Madaan et al., 2023; Shinn et al., 2023). Tool-integrated reasoning systems that couple symbolic solvers with neural generation (Lyu et al., 2023; Lu et al., 2024) improve single-turn accuracy, with Lyu et al. (2023) demonstrating that removing the deterministic external solver causes a 50-point accuracy drop on GSM8K. However, aggregate gains can obscure shifts in the composition of residual errors. Endpoint accuracy may improve substantially even as assignment-level drift remains unchanged or worsens, because the error types that repair eliminates are not necessarily the ones most visible to users. Recent work on the limits of LLM self-verification reaches a related conclusion. Stechly et al. (2025) show that when GPT-4 is tasked with both generating and critiquing its own answers, performance actually decreases, and that substantial gains require a sound external verifier regardless of critique richness. Our analysis extends this concern to interactive trajectories by decomposing residuals by operational failure type.

Formal methods in neural systems. Satisfiability solving and minimal unsatisfiable subset extraction are well-established tools in symbolic debugging and verification (de Moura & Bjørner, 2008; Liffiton & Sakallah, 2008; Belov et al., 2012; Biere et al., 2009). A separate but related thread comes from task-oriented dialogue, where belief state updates track evolving user requirements (Young et al., 2013; Wu et al., 2019). The ledger mechanism in our system draws on both traditions. It maintains formal constraint sets, as in symbolic verification, but updates them incrementally at each conversational turn, as in dialogue state tracking. Our contribution is adapting this combined toolbox to neural multi-turn traces through fixed turn-level solver instrumentation, trigger-conditioned repair routing, and paired inferential analysis over interactive trajectories.

3 METHOD

3.1 NOTATION AND STATE SEMANTICS

The multi-turn setting requires distinguishing between the raw model output and the structured state derived from it. We write u_t for the user message at turn t, a_t for the model's response text, and A_t for the structured assignment parsed from a_t. The cumulative gold constraints are denoted by $\mathcal{C}_{1:t}$, extracted constraints by $\widehat{\mathcal{C}}_t$ (the model's parse of new constraints at turn t), and ledger state by L_t. The predicate $\mathrm{SAT}(\cdot)$ indicates solver satisfiability; we write $\mathrm{UNSAT}(\cdot)$ for its negation.

Each problem is a turn sequence $\{u_t\}_{t=1}^{T}$ with cumulative gold constraints

$$\mathcal{C}_{1:t} = \bigcup_{\tau=1}^{t} \mathcal{C}_\tau^{\text{new}}.$$

Turn-level correctness is defined by constraint satisfaction rather than string match against a single witness assignment. The operational correctness predicate is

$$\mathrm{Correct}(A_t) = \mathrm{Parse}(A_t) \ \wedge \ \mathrm{Complete}(A_t) \ \wedge \ \mathrm{Satisfies}(A_t, \mathcal{C}_{1:t}).$$

In implementation, `answer_correct` is obtained by checking satisfiability of $\mathcal{C}_{1:t}$ with A_t injected as an assignment in Z3. This definition remains valid when multiple satisfying assignments exist. The same predicate appears in drift diagnostics with L_t replacing $\mathcal{C}_{1:t}$; the constraint set argument determines which notion of consistency is tested. We measure accuracy against gold cumulative constraints $\mathcal{C}_{1:t}$, while drift is a diagnostic defined against the model-maintained ledger L_t.

The distinction between ledger satisfiability and assignment validity is central to the paper. A turn can preserve $\mathrm{SAT}(L_t)$ while still violating active commitments through $\neg\mathrm{Satisfies}(A_t, L_t)$. This separation allows contradiction and drift to be measured as distinct channels rather than merged into a single error indicator. Formally, let $\Phi(A_t)$ denote the assignment constraints induced by the parsed answer. Then

$$\mathrm{Satisfies}(A_t, S) = \mathrm{SAT}(S \cup \Phi(A_t)).$$

The parser predicate $\mathrm{Parse}(A_t)$ is one only when the response is valid schema-conforming JSON for the domain. The completeness predicate $\mathrm{Complete}(A_t)$ is one only when each required entity is assigned exactly once. The ledger update is

$$\mathrm{Merge}(L_{t-1}, \widehat{\mathcal{C}}_t) = L_{t-1} \cup \mathrm{Dedup}(\widehat{\mathcal{C}}_t),$$

where Dedup removes canonical duplicates before insertion.

These predicates partition turn outcomes into three categories. A turn is *consistent* when the ledger is satisfiable and the assignment respects it. When the ledger remains satisfiable but the assignment violates it, the turn exhibits *drift*. When the ledger itself becomes unsatisfiable, the turn exhibits *contradiction*. The critical distinction is that drift produces no solver alarm, making it invisible to any system that checks only state consistency. Figure 2 illustrates all three outcomes on a four-turn scheduling trajectory where drift occurs at the final turn.

3.2 SYSTEM COMPONENTS

The evaluation system decomposes each turn into four stages, reflecting a deliberate separation of generation from verification. A generator G produces the response a_t given the current user message and prior ledger state. An extractor E then parses the response alongside the user message to identify newly introduced constraints $\widehat{\mathcal{C}}_t$. These feed into a verifier V, which runs both solver-level satisfiability checks and policy-level checks on the parsed assignment. Finally, a repair policy R examines the verifier output and decides whether to issue a retry with targeted feedback.

The policy checks validate three properties: whether the response parses as valid JSON, whether the assignment covers all required entities, and whether the assignment satisfies the active ledger constraints. Each check emits a deterministic trigger code, a fixed label that identifies the failure type. These codes serve a dual purpose: at runtime they route the repair decision, and post-hoc they enable fine-grained failure decomposition. Table 2 lists all five trigger codes and their formal conditions; Algorithm 1 shows how they integrate into the turn processing loop.

We evaluate four inference policies on this shared infrastructure: DIRECT, CHAIN-OF-THOUGHT, LEDGER, and MUS-REPAIR. Figure 2 compares their formal signatures on a scheduling example where drift occurs at the final turn. The repair step is active only for MUS-REPAIR and only when the verifier emits one or more failing triggers. Crucially, we hold the extractor and verifier fixed across all methods so that observed differences reflect reasoning and repair strategy, not variation in parsing or verification logic.

Algorithm 1 Turn processing with verification and optional repair.

Input: u_t, L_{t-1}, method m, turn t
Output: response a'_t, ledger L_t

1: $a_t \leftarrow G(u_t, L_{t-1})$
2: $\widehat{\mathcal{C}}_t \leftarrow E(u_t, a_t, t)$
3: $L_t \leftarrow L_{t-1} \cup \mathrm{Dedup}(\widehat{\mathcal{C}}_t)$
4: $(\mathrm{sat}_t,\ \mathcal{T}_t) \leftarrow V(L_t, a_t)$
5: **if** $m \neq$ MUS-REPAIR **or** $(\mathrm{sat}_t \ \wedge\ \mathcal{T}_t = \emptyset)$ **then**
6: **return** (a_t, L_t)
7: **end if**
8: **if** $\neg\,\mathrm{sat}_t$ **then**
9: $\mathcal{U}_t \leftarrow \mathrm{MUS}(L_t)$
10: **else** $\mathcal{U}_t \leftarrow \emptyset$
11: **end if**
12: $a'_t \leftarrow R\big(u_t, L_{t-1}, \mathrm{Render}(\mathcal{T}_t, \mathcal{U}_t)\big)$
13: $L_t \leftarrow L_{t-1} \cup \mathrm{Dedup}\big(E(u_t, a'_t, t)\big)$
14: **return** (a'_t, L_t)

Table 2: Trigger codes emitted by the verifier V. Each code maps to a specific failure channel.

Trigger code	Condition
Answer-Ledger Conflict	$\mathrm{SAT}(L_t) \wedge \neg\mathrm{Sat}(A_t, L_t)$
Unsatisfiable Ledger	$\neg\mathrm{SAT}(L_t)$
Incomplete Assignment	$\neg\mathrm{Complete}(A_t)$
Answer Parse Failure	$\neg\mathrm{Parse}(A_t)$
Extraction Failure	$\widehat{\mathcal{C}}_t = \emptyset$

Lines 1–4 run the forward pass. If no triggers fire, the turn returns without repair (line 6). Otherwise, MUS is computed for contradiction or set to $\emptyset$ for drift (lines 8–10), and the repair policy retries with $(\mathcal{T}_t, \mathcal{U}_t)$.

3.3 MINIMAL UNSATISFIABLE SUBSET FOR REPAIR

When V detects an unsatisfiable ledger state, MUS-REPAIR computes a minimal unsatisfiable subset $\mathcal{U}_t \subseteq L_t$ such that

$$\mathrm{UNSAT}(\mathcal{U}_t) = 1, \quad \forall \mathcal{U}' \subset \mathcal{U}_t,\ \mathrm{SAT}(\mathcal{U}') = 1.$$

This subset is minimal in set inclusion and identifies the smallest committed constraint set that is jointly inconsistent at turn t. The retry prompt receives trigger diagnostics and, for unsatisfiable states, the corresponding $\mathcal{U}_t$. The same retry channel is used for satisfiable assignment failures through policy triggers, so contradiction and drift are handled in one controlled repair interface. The repair feedback packet is

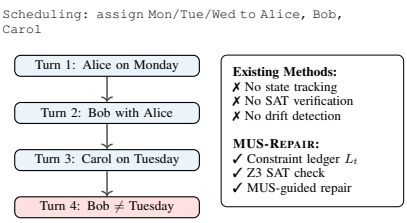
$$F_t = \begin{cases} (\mathcal{T}_t, \mathcal{U}_t), & \mathrm{UNSAT}(L_t) = 1, \\ (\mathcal{T}_t, \emptyset), & \mathrm{SAT}(L_t) = 1 \ \wedge\ \mathcal{T}_t \neq \emptyset. \end{cases}$$

MUS is injected only for contradiction events, while satisfiable assignment failures are repaired with policy diagnostics and the prior ledger state.

3.4 FAILURE CHANNEL DECOMPOSITION

The conceptual contradiction indicator is $I_t^{\mathrm{unsat}} = \mathbf{1}[\mathrm{UNSAT}(L_t)]$. The conceptual drift indicator is $I_t^{\mathrm{drift}} = \mathbf{1}[\mathrm{SAT}(L_t) \wedge \neg\mathrm{Satisfies}(A_t, L_t)]$. Current logs provide direct contradiction status and trigger-level diagnostics on MUS-REPAIR traces. We measure contradiction with `z3_sat`=0 and drift with the Answer-Ledger Conflict trigger `answer_ledger_conflict`, which corresponds to a satisfiable ledger with a violating assignment. Parser and completeness failures are tracked by Answer Parse Failure, Incomplete Assignment, and Constraint Extraction Failure triggers.

Primary reporting uses turn-level accuracy as defined by $\mathrm{Correct}(A_t)$. The inference protocol is described in Section 4.3.



Direct (✗)

$$G : H_{1:t} \times u_t \to A_t$$

Chain-of-Thought (✗)

$$G : H_{1:t} \times u_t \to \{r_k\}_{k=1}^{K},\ A_t$$

Ledger (✗)

$$G : H_{1:t} \times u_t \times L_{t-1} \to A_t$$
$$L_t \leftarrow L_{t-1} \cup E(u_t, a_t)$$

MUS-Repair (✓)

$$G : H_{1:t} \times u_t \times L_{t-1} \to A_t$$
$$V : \mathrm{SAT}(L_t) \wedge \mathrm{Satisfies}(A_t, L_t)$$
$$R : (\mathcal{T}_t, \mathcal{U}_t) \to A'_t$$

Figure 2: Comparison of constraint reasoning approaches. Left: a four-turn scheduling trajectory where drift occurs at turn 4. Center: properties of baselines vs. MUS-Repair. Right: formal method signatures with orange highlighting implicit context accumulation and teal highlighting explicit ledger state and solver verification. Only MUS-Repair detects the drift because it verifies both $\mathrm{SAT}(L_t)$ and $\mathrm{Satisfies}(A_t, L_t)$.

4 Experimental Setup

4.1 Benchmark Construction

Benchmark problems are generated by a procedure that guarantees every gold interaction trajectory is satisfiable at each turn. For each domain $\mathcal{D} \in \{\texttt{logic\_grid}, \texttt{scheduling}, \texttt{seating}\}$, the generator samples entities, contextual framing, and one to three candidate constraints per turn. It accepts a candidate set only when the cumulative set remains satisfiable under Z3

$$\mathrm{SAT}\left(\mathcal{C}_{1:t-1} \cup \widehat{\mathcal{C}}_t^{\text{cand}}\right) = 1.$$

If the candidate set is unsatisfiable, the turn is resampled until acceptance or retry budget exhaustion. This process ensures that every gold interaction trajectory is satisfiable at each turn. Generation also removes duplicate constraints by canonical form before satisfiability checks, which prevents trivial repetition across turns.

Gold correctness does not assume a unique assignment. We verify by checking satisfiability of cumulative constraints conjoined with the parsed answer assignment, so a response is correct whenever it satisfies the active constraints, even if multiple assignments are valid.

Each domain uses a fixed template that determines the structural parameters of generated problems. Logic-grid instances pair four entities with three categorical attributes, producing compact but combinatorially rich assignments. Scheduling instances involve five to seven events assigned to temporal slots, with predicates such as ordering and simultaneity constraints. Seating instances are the most spatially complex, placing six to eight participants around round or rectangular tables subject to adjacency, separation, and positional constraints. Turn count is sampled between four and ten, with one to three new constraints introduced per turn.

The final corpus has 1,020 problems with a fixed seed split of 816 test and 204 development instances. Table 3 summarizes structural properties by domain; the Final column is the mean number of cumulative active constraints at the last turn.

Table 3: Benchmark structure by domain.

Domain	Split	Turns [min,max]	Ent.	Vocab	Final
Logic-Grid	272/68/340	6.89 [4,10]	4.00	4	11.57
Scheduling	272/68/340	7.06 [4,10]	5.92	6	12.83
Seating	272/68/340	6.97 [4,10]	7.01	7	11.20

4.2 Evaluation Stack and Model Matrix

All methods run in a shared OpenAI-compatible serving stack with identical extraction, verification, and logging paths. The model matrix contains Qwen3-8B, Qwen3-32B, gpt-oss-20b, and gpt-oss-120b. The Qwen models are from the Qwen3 family (Qwen Team, 2025). The gpt-oss models are OpenAI's open-weight releases at 20B and 120B parameters, served locally through vLLM under the same stack as the Qwen runs. For gpt-oss evaluations we use deterministic decoding with temperature

set to zero and default reasoning configuration, matching the same deterministic setting used for paired comparisons in the Qwen runs.

Each setting evaluates all four methods on the full test split

$$5{,}672 \text{ rows per method per model} \times 4 \text{ methods} \times 4 \text{ models} = 90{,}752 \text{ turn evaluations.}$$

The gpt-oss-120b run is complete for all methods at 5,672 rows and 816 problems per method and is included in all main-text tables.

4.3 Inference Protocol and Robustness Checks

To ensure that reported accuracy differences are not artifacts of problem sampling, we construct 95% bootstrap confidence intervals at the problem level and assess pairwise significance using sign-permutation tests against DIRECT. We then apply Benjamini-Hochberg correction across all reported comparisons to control the false-discovery rate.

Prompt templates, JSON schema constraints, repair message format, and extraction prompts are documented in Appendix A. Runtime controls are fixed across methods with temperature set to zero, maximum repair attempts set to two, maximum truncation retries set to two, and ledger token budget set to 3,000 unless a serving-side safety clamp is required.

One practical consideration is that the gpt-oss models occasionally produce truncated responses, which could in principle affect accuracy estimates. We verified that restricting analysis to non-truncated responses preserves both the method ordering and the MUS-Repair margins, indicating that truncation is not a systematic confound.

5 Results

5.1 Primary Accuracy

Table 4 presents the full results. **MUS-REPAIR is the strongest method in every model setting**, with gains over DIRECT ranging from $+2.0$ pp on Qwen3-8B to $+16.2$ pp on gpt-oss-20b. Every MUS-Repair comparison survives paired problem-level permutation tests after Benjamini-Hochberg correction ($q_{\mathrm{FDR}} < 0.03$ in all cases), and the pattern holds when tested against each model's strongest non-MUS comparator rather than DIRECT alone (Appendix Table 10). Structured repair helps the weakest model (Qwen3-8B, 30%) and the strongest (gpt-oss-20b, 69%) alike. This consistency across a wide capability range suggests that the benefit comes from the verification-and-retry mechanism itself rather than from model-specific artifacts. The other methods show mixed results. Ledger significantly hurts Qwen3-8B (-3.1 pp, $q = 0.003$) and gpt-oss-120b (-2.3 pp, $q = 0.022$), while CoT produces modest gains on Qwen3-32B and gpt-oss-120b but not on the other two models.

5.2 Capability Scaling of Repair Gains

Stronger models benefit more from structured repair. Raw accuracy gains are larger for more capable models, but comparing absolute improvements across models with different baselines can be misleading. A more informative measure is relative lift, which normalizes the MUS-REPAIR gain by each model's best non-MUS baseline.

$$\rho_m = \frac{A_m^{\mathrm{MUS}} - \max\left(A_m^{\mathrm{Direct}}, A_m^{\mathrm{CoT}}, A_m^{\mathrm{Ledger}}\right)}{\max\left(A_m^{\mathrm{Direct}}, A_m^{\mathrm{CoT}}, A_m^{\mathrm{Ledger}}\right)}.$$

Relative lift rises from 6.4% on Qwen3-8B to 27.9% on gpt-oss-20b before dropping to 16.2% on gpt-oss-120b. The non-monotonic drop at gpt-oss-120b resists simple scaling predictions, but the overall trajectory still shows that repair remains materially beneficial even at the highest capability level tested. One plausible explanation centers on instruction-following fidelity. The repair signal is a structured prompt containing trigger codes and a minimal unsatisfiable subset. Converting this signal into a corrected assignment requires precisely the kind of structured instruction following that improves with model capability. A model that cannot parse the signal treats the retry as noise.

Table 4: Turn-level accuracy, paired inferential tests versus DIRECT ($n = 816$), and depth retention. Highlighted rows show MUS-REPAIR. Retain % = turn-10 accuracy / turn-1 accuracy, measuring how well each method preserves performance as constraints accumulate.

Model	Method	Acc. (%)	Δ (pp)	95% CI (pp)	q_{FDR}	Retain %
Qwen3-8B	Direct (baseline)	28.19	—	—	—	5.1
Qwen3-8B	Chain-of-Thought	27.91	−0.19	[−2.01, +1.63]	0.8422	6.7
Qwen3-8B	Ledger	25.23	−3.14	[−4.99, −1.31]	0.0034	5.7
Qwen3-8B	MUS-Repair	**30.01**	2.03	[+0.32, +3.71]	0.0295	**8.8**
Qwen3-32B	Direct (baseline)	28.93	—	—	—	6.5
Qwen3-32B	Chain-of-Thought	31.44	2.54	[+0.73, +4.38]	0.0187	2.5
Qwen3-32B	Ledger	31.44	1.63	[−0.93, +4.20]	0.2204	23.6
Qwen3-32B	MUS-Repair	**38.22**	9.03	[+6.98, +11.00]	0.0002	**15.7**
gpt-oss-20b	Direct (baseline)	51.80	—	—	—	23.1
gpt-oss-20b	Chain-of-Thought	50.35	−1.39	[−3.05, +0.27]	0.1183	19.4
gpt-oss-20b	Ledger	53.70	1.91	[+0.25, +3.60]	0.0327	29.1
gpt-oss-20b	MUS-Repair	**68.71**	16.20	[+14.52, +17.90]	0.0002	**48.3**
gpt-oss-120b	Direct (baseline)	52.12	—	—	—	30.2
gpt-oss-120b	Chain-of-Thought	53.95	2.04	[+0.36, +3.70]	0.0295	24.2
gpt-oss-120b	Ledger	50.02	−2.29	[−4.03, −0.59]	0.0220	36.4
gpt-oss-120b	MUS-Repair	**62.68**	10.05	[+8.40, +11.72]	0.0002	**42.9**

Ledger-only tracking, by contrast, is not uniformly positive. It hurts Qwen3-8B (−3.0 pp), is near-neutral on Qwen3-32B, helps gpt-oss-20b (+1.9 pp), and drops again on gpt-oss-120b (−2.1 pp). Explicit state tracking trades control benefits against the cost of occupying prompt context, and the balance shifts with model capability (Section 7).

5.3 DEPTH DEGRADATION

Every model shows steep accuracy decline with turn depth (Figure 3). The decline is dramatic across the full model range. Qwen3-8B drops from 72% at turn one to 6% at turn ten under MUS-REPAIR, and even gpt-oss-120b falls from 93% to 40% over the same span. Crucially, the shape of the decline is steep rather than gradual, consistent with the probability of violating at least one constraint growing combinatorially as the active set expands. Higher capability lifts the entire curve but does not flatten it. This pattern is reminiscent of the positional sensitivity documented by Liu et al. (2024) for long contexts, but here the degradation is temporal rather than positional. Long-horizon state maintenance appears to be a qualitatively harder problem that will likely require architectural support beyond pure scaling.

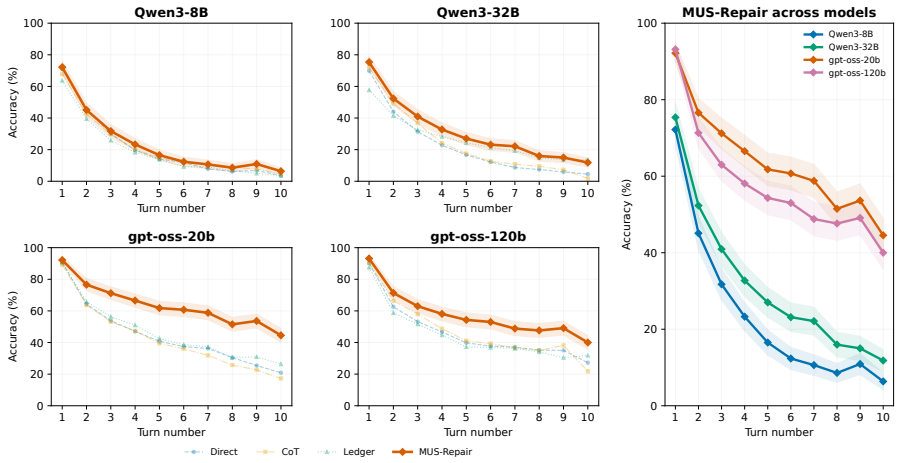


Figure 3: Per-turn accuracy curves for all models and methods. Higher capability lifts the curve but does not flatten it.

5.4 Domain Structure

Seating is the hardest domain and scheduling the easiest across all models (Table 5). This ranking holds consistently from the smallest Qwen model to the largest gpt-oss run, indicating that the difficulty is inherent in the constraint topology rather than an artifact of any single model's weaknesses. Seating problems involve circular positional constraints (adjacency, separation, wrapping) that require globally consistent placement of 6–8 entities, whereas scheduling admits more localized solutions. The gap is largest on gpt-oss-20b, where scheduling reaches 85.7% while seating remains at 38.7%.

Table 5: MUS-Repair domain-conditioned accuracy (%). Scheduling is consistently easiest and seating hardest.

Model	Logic-Grid (%)	Scheduling (%)	Seating (%)
Qwen3-8B	43.1	34.7	12.1
Qwen3-32B	43.9	55.7	14.8
gpt-oss-20b	81.4	85.7	38.7
gpt-oss-120b	64.2	87.2	36.3

6 Failure Analysis

6.1 Contradiction and Drift Decomposition

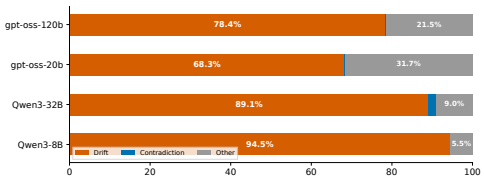
Because MUS-Repair logs solver status and trigger codes at every retry, we can decompose residual errors into three distinct channels: contradiction, in which the ledger becomes unsatisfiable; drift, in which the ledger remains satisfiable but the assignment violates it; and formatting or extraction errors. We measure contradiction by `z3_sat`=0 and drift by the Answer-Ledger Conflict trigger, which fires when a satisfiable ledger accompanies a returned assignment that violates active constraints.

Contradiction repair does not remove the dominant residual failure mode. This is the paper's central empirical finding. Drift accounts for 68.3% to 94.5% of repair-attempted rows across all settings (Table 6), while contradiction drops to 0.0–1.9%, with near-zero counts in the gpt-oss runs. The magnitude of this imbalance is best appreciated through raw counts. On Qwen3-8B, 3,970 of 4,201 repair-attempted rows end in drift, with zero contradiction events. On Qwen3-32B, which triggers unsatisfiable states more frequently due to aggressive constraint extraction, contradiction still accounts for only 72 of 3,858 rows (1.9%). The remaining errors, categorized as "Other," encompass parse failures, incomplete assignments, and extraction errors. These are comparatively rare and model-dependent. On gpt-oss-20b they are dominated by format-compliance issues (664 of 823 rows), while on Qwen3-8B they account for just 5.5% of repair-attempted rows. The practical implication is that reducing unsatisfiable ledgers is necessary for reliability, but most remaining user-facing errors stem from assignments that violate a satisfiable maintained state.

Table 6: MUS-Repair failure channel decomposition over repair-attempted rows.

Model	Drift	UNSAT	Other	Drift (%)	UNSAT (%)	Other (%)
Qwen3-8B	3970	0	231	**94.5**	0.0	5.5
Qwen3-32B	3438	72	348	**89.1**	1.9	9.0
gpt-oss-20b	1774	1	823	**68.3**	0.0	31.7
gpt-oss-120b	2115	2	580	**78.4**	0.1	21.5

Figure 4: Residual failure decomposition after MUS-Repair by model. Drift dominates; contradiction is near-invisible.



6.2 Trigger Composition, Repair Outcomes, and Residual Overlap

Trigger event counts (Table 7) reinforce the decomposition. Answer-Ledger Conflict dominates every model, firing 12,089 times on Qwen3-8B and 5,300 times on gpt-oss-20b. Unsatisfiable Ledger, by contrast, concentrates in Qwen3-32B (212 events) and stays scarce elsewhere ($\leq$4 events on the gpt-oss models). Note that Table 6 counts final-row outcomes while Table 7 counts trigger events across retries, so magnitudes differ by construction.

Not all trigger types respond equally to repair. Schema-completion failures, such as missing entities or malformed JSON, are relatively straightforward to fix with a retry prompt because the error is localized and well-defined. Assignment-level consistency failures, by contrast, require the model to simultaneously satisfy all active constraints while revising its answer, a much harder task. The data bear this out. On Qwen3-8B, post-repair accuracy reaches 65.5% for Incomplete Assignment triggers but only 4.0% for Answer-Ledger Conflict triggers. On Qwen3-32B, the same contrast is 69.8% versus 5.5%. This recoverability gap explains why drift persists as the dominant failure mode even after multiple repair attempts.

Table 7: Trigger event counts in MUS-REPAIR traces by model. Answer-Ledger Conflict dominates in every setting, while Unsatisfiable Ledger is concentrated in Qwen3-32B.

Trigger	Qwen3-8B	Qwen3-32B	gpt-oss-20b	gpt-oss-120b
Answer-Ledger Conflict	12 089	10 489	5300	6255
Incomplete Assignment	218	528	1036	31
Answer Parse Failure	0	2	664	810
Constraint Extraction Failure	0	2	559	36
Unsatisfiable Ledger	1	212	2	4

Residual error overlap across models is high rather than fragmented. Qwen3-8B covers 95.0% of gpt-oss-20b residual MUS-REPAIR errors, and Qwen3-32B covers 92.3%. This shared residual set points to common hard regions of the benchmark rather than disjoint model-specific failure pockets. The problems that resist repair for one model tend to resist it for all of them, which suggests the difficulty is intrinsic to the constraint structure rather than tied to any particular model's weaknesses.

7 DISCUSSION

The central finding is not that MUS-Repair works, but what it leaves behind. After contradiction-aware repair, the residual error mass concentrates in satisfiable drift rather than in unsatisfiable states. This asymmetry has consequences for system design, scaling expectations, and evaluation methodology.

Drift dominance in deployed systems. A system that ships satisfiability checks as its primary reliability gate will miss the majority of user-visible failures in this benchmark family. The failure mode is insidious because it evades every standard check. The internal ledger remains consistent, the solver raises no alarm, and the returned answer nonetheless violates a commitment the user already accepted. Unlike contradiction, which at least signals that something has gone wrong, drift produces confident answers that pass every automated check inspecting only state consistency. For scheduling or resource allocation assistants, this means a user who asks "remind me of the constraints so far" receives a valid summary while the assignment silently breaks one of those same constraints. Detecting this class of error requires a second verification layer that explicitly checks the assignment against the maintained state. This architectural requirement parallels the finding of Stechly et al. (2025) that sound external verification is necessary regardless of critique sophistication.

Why stronger models benefit more from symbolic feedback. Relative MUS-Repair lift rises from 6.4% on Qwen3-8B to 27.9% on gpt-oss-20b. We see two complementary mechanisms at work, both supported by the trigger data. The first is baseline error composition. On Qwen3-8B, Answer-Ledger Conflict accounts for 12,089 of 12,308 total triggers (98%), meaning nearly all repair attempts target drift, a failure type that resists retry. On gpt-oss-20b, the trigger mix is more diverse (5,300 drift, 1,036 incomplete, 664 parse), giving the repair loop a broader surface of recoverable errors. The second mechanism is instruction-following fidelity. The repair signal is a structured prompt containing trigger codes, violated constraints, and a minimal unsatisfiable subset. Converting this signal into a corrected assignment requires the kind of structured instruction following that improves with model capability; a model that cannot parse the signal treats the retry as noise. The trigger data bear this out indirectly. Post-repair accuracy on Answer-Ledger Conflict triggers rises from 4.0% on Qwen3-8B to 33.3% on gpt-oss-20b, a factor-of-eight improvement, suggesting that the larger model is better at acting on the structured feedback even for the hardest failure type. The non-monotonic drop at gpt-oss-120b complicates this picture. One possibility is that the largest

model's implicit state tracking is already strong enough that the marginal value of explicit MUS feedback diminishes, even though absolute accuracy still improves.

Depth collapse as accumulation. The depth curves in Figure 3 present perhaps the most challenging finding for scaling-based solutions. The steep decline, rather than gradual erosion, suggests that each new constraint does not simply add a fixed probability of error. Instead, the probability of violating at least one constraint grows combinatorially with the active set, creating a qualitatively harder problem at each successive turn. This framing suggests that flattening the depth curve will require mechanisms that scale sublinearly with constraint count, such as hierarchical state abstractions or incremental re-verification, rather than relying on raw model capacity alone.

Ledger tracking and context competition. Explicit ledger injection helps gpt-oss-20b but hurts Qwen3-8B, and the benefit declines again at gpt-oss-120b. The likely mechanism is context competition. Serializing the ledger (up to 3,000 tokens) consumes prompt budget that a smaller model needs for reasoning. A larger model absorbs the overhead and uses the explicit state productively. The renewed decline at gpt-oss-120b suggests that above a capability threshold the model's implicit tracking is competitive with explicit injection, so the added context cost outweighs the control benefit.

Toward drift-targeted repair. The current repair loop is contradiction-oriented, identifying minimal unsatisfiable subsets and feeding them back. Drift, by contrast, receives only generic policy diagnostics without localizing which constraints are violated or which entities are misplaced. Closing this gap requires localizing the violated constraints.

$$\mathcal{V}_t = \big\{ c \in L_t : \neg\mathrm{SAT}\big(\{c\} \cup \Phi(A_t)\big) \big\},$$

which mirrors the MUS definition structurally: MUS localizes the source of contradiction within the ledger, while $\mathcal{V}_t$ localizes the source of drift within the assignment.

Beyond solver-structured domains. Our benchmark uses formal constraint sets because they enable sound verification, but satisfiable drift is not specific to constraint satisfaction. Any multi-turn system that maintains evolving commitments can exhibit it. A travel-planning assistant might confirm "no flights on Sunday" and later propose a Sunday itinerary; a code-editing agent might acknowledge a variable rename and subsequently reference the old name. In these open-domain settings, detecting drift would require extracting implicit constraints from natural language, likely through entailment-based commitment tracking rather than SAT solving. The core diagnostic question remains the same: is the system's internal state valid, and does the output respect that state? We expect drift to dominate in open-domain settings as well, because the underlying cause, forgetting prior commitments while maintaining a coherent narrative, is a property of how language models process sequential context rather than an artifact of the constraint format.

Evaluation implications. The distinction between state-level and assignment-level failure extends beyond our benchmark. Dialogue state trackers, collaborative document editors, and iterative code generators all maintain evolving commitments across turns. We suggest that multi-turn evaluations in these domains similarly decompose errors by whether the system's internal state became invalid or whether the output simply failed to respect a valid state. Reporting the two channels separately would prevent the pattern we document here, where progress on one failure type masks stagnation on the other, and would give practitioners a clearer picture of where reliability investments should be directed.

Taken together, these results argue that contradiction detection, though necessary, is not sufficient for reliable multi-turn systems. A second verification layer that checks the returned assignment against the maintained state is needed to catch the dominant failure mode. Reporting contradiction and drift as separate evaluation channels, rather than merging them into a single accuracy number, would give practitioners a clearer picture of where residual risk concentrates.

8 LIMITATIONS

Several scope limitations bear on the generalizability of our findings. The study evaluates four open-weight models from two families but does not include closed-weight frontier systems or specialist fine-tuned variants, either of which might exhibit different drift-to-contradiction ratios. Failure-channel decomposition relies on the solver-state and trigger logs that MUS-REPAIR produces at each retry. We do not have equivalent per-turn logging for the non-repair methods, which limits fully symmetric cross-method comparison of failure channels. Post-hoc instrumentation of non-repair traces with the same solver checks would enable symmetric decomposition; we leave this to follow-up work. The benchmark covers three solver-structured domains, and whether the drift-dominance finding transfers to open-domain dialogue, where constraints are implicit and verification is harder, remains an open question. Finally, we evaluate a single repair routing design without ablating over trigger definitions, retry budgets, or alternative repair controllers. Different routing policies might shift the balance between contradiction and drift in the residual error distribution.

9 CONCLUSION

This paper introduced a solver-instrumented multi-turn benchmark that cleanly separates two failure modes, contradiction and satisfiable drift, and used it to evaluate four reasoning methods across four open-weight models. MUS-Repair produces significant gains in every setting after false-discovery correction, but the errors that survive are overwhelmingly drift. Models rarely contradict themselves after structured feedback, but they still forget prior commitments. This forgetting compounds with conversational depth, and accuracy declines steeply even on the strongest model, suggesting that long-horizon state maintenance remains an open challenge regardless of scale.

These findings point to a concrete gap in current evaluation practice. Solver-level contradiction checks are necessary but insufficient. Reliable multi-turn systems must also validate that the returned assignment respects the maintained state. Reporting contradiction and drift as separate channels, rather than merging them into a single accuracy number, exposes where the real residual risk lies.

ACKNOWLEDGEMENTS

Formatting acknowledgement. The presentation style of this paper, including color-coded table cells, minipage layouts, enumerated findings, and caption formatting, was adapted from the ERRORRADAR benchmark paper (Yan et al., 2025).

REFERENCES

Anton Belov, Ines Lynce, and João Marques-Silva. Muser2: An efficient MUS extractor. In *Theory and Applications of Satisfiability Testing*, 2012.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

Wenhu Chen, Xuezhe Ma, Xueguang Wang, William Cohen, et al. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

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.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2023.

Dong Han, Jiaxin Cao, Weijia Zheng, Bill Lin, Dawn Song, and Juan Carlos Niebles. Verifiagent: A multi-tool agentic system for verifying factual claims. *arXiv preprint arXiv:2504.00406*, 2025.

Hailuo Hu, Ming Li, Feng Jiang, Hongyuan Li, et al. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.03549*, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yutaka Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Nelson Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2024.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023.

Aman Madaan, Niket Tandon, Peter Clark, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2022.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Oskar Stechly, Sercan O. Arik, Tomas Pfister, and Hanlin Goh. On the limitations of verifier-based self-verification in LLM reasoning. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2019.

Yibo Yan, Shen Wang, Jiahao Huo, Hang Li, Boyan Li, Jiamin Su, Xiong Gao, Yi-Fan Zhang, Tianlong Xu, Zhendong Chu, Aoxiao Zhong, Kun Wang, Hui Xiong, Philip S. Yu, Xuming Hu, and Qingsong Wen. ErrorRadar: Benchmarking complex mathematical reasoning of multimodal large language models via error detection. In *ICLR 2025 Workshop on Reasoning and Planning for LLMs*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023a.

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

Shunyu Yao, Howard Peng, Yann Koh, Peter Washington, and Karthik Narasimhan. COLLIE: Systematic construction of constrained text generation tasks. In *International Conference on Learning Representations*, 2024.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D. Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

A Appendix Prompt and Policy Artifacts

This appendix documents the exact prompt text and templates used by the execution pipeline.

A.1 Main-Turn System Prompts

System Prompt `direct`

You solve multi-turn logical constraint satisfaction problems. Track all prior commitments and keep the final assignment consistent with every active constraint. Return only the final JSON solution.

System Prompt `cot`

You solve multi-turn logical constraint satisfaction problems. Track all prior commitments and reason briefly before answering. Output at most 3 short bullets, then the final JSON solution.

System Prompt `ledger_only`

You solve a multi-turn logical constraint problem with an explicit ledger. Treat the ledger as committed state and keep your answer consistent with it and the new turn constraints. Return only JSON.

System Prompt `mus_repair`

You solve a multi-turn logical constraint problem with formal consistency checks and targeted repair signals. Use the ledger as committed state, address any repair signal directly, and provide a complete consistent solution. Return only JSON.

A.2 Additional System Prompts for Extraction and Retries

Answer Retry System Prompt

You are a strict JSON formatter. Output one valid JSON object only. No markdown or prose.

Constraint Extraction System Prompt

You extract formal constraints from an assistant answer. Extract only constraints introduced in the latest user turn. Do not restate full solution assignments unless they directly encode one allowed constraint.

A.3 Main User Message Assembly Template

User Prompt Assembly Template

Problem setup (first turn only): include `Domain: <domain>` and `Entities: <entity list>` together with domain-specific context such as seat labels, time slots, or logic-grid categories.
State block (ledger methods only): include `Current ledger: <serialized constraints>`.
Current turn block: include `New constraints from user: <latest user message>`, optional `Repair signal: <trigger codes + optional MUS subset>`, and a domain-specific JSON schema hint.

A.4 Repair Signal Template

Repair Signal Format

REPAIR REQUIRED Detected issue classes are provided as `<trigger code> : <detail>`. An optional MUS subset is provided as `<constraint id> : "<constraint text>" (turn <t>)`. The required action is to return a revised JSON solution that resolves all listed issues.

A.5 Constraint Extraction Prompt Template

Constraint Extraction Prompt

System instruction: extract only constraints introduced in the latest user turn from the assistant answer. **User block fields:** include `Domain`, `Source turn`, and `Entities`; include `Latest user message` and `Assistant response`; specify the domain-specific allowed constraint vocabulary; enforce strict rules on type names, argument order, and turn locality; and require JSON output with key `"constraints"`.

A.6 Retry Prompt Templates

Answer Reformat Retry Prompt

System: strict JSON formatter, one valid JSON object only.
User: reformat the response as JSON only; preserve required entities as keys and add no commentary.

Truncation Retry Prompt

Previous answer was clipped by token limit. Retry with one compact JSON object only: no bullets, no prose, no analysis.

B Appendix Supplementary Tables

Table 8: Domain-conditioned turn-level accuracy.

Model	Domain	Direct (%)	CoT (%)	Ledger (%)	MUS-Repair (%)
Qwen3-8B	Logic-Grid	39.7	37.5	36.2	43.1
Qwen3-8B	Scheduling	35.0	35.1	29.1	34.7
Qwen3-8B	Seating	9.8	11.0	10.3	12.1
Qwen3-32B	Logic-Grid	35.6	38.6	24.0	43.9
Qwen3-32B	Scheduling	41.1	44.8	56.9	55.7
Qwen3-32B	Seating	9.9	10.7	13.0	14.8
gpt-oss-20b	Logic-Grid	57.3	56.1	60.8	81.4
gpt-oss-20b	Scheduling	64.4	64.6	68.8	85.7
gpt-oss-20b	Seating	33.5	30.1	31.2	38.7
gpt-oss-120b	Logic-Grid	53.6	56.2	49.1	64.2
gpt-oss-120b	Scheduling	73.7	72.3	72.8	87.2
gpt-oss-120b	Seating	28.7	33.1	27.9	36.3

Table 9: Truncation robustness check.

Model	Method	Trunc. (%)	Acc. All (%)	Acc. Non-trunc. (%)
gpt-oss-120b	Direct	0.37	52.1	52.3
gpt-oss-120b	Chain-of-Thought	0.23	53.9	54.1
gpt-oss-120b	Ledger	0.14	50.0	50.1
gpt-oss-120b	MUS-Repair	0.11	62.7	62.7
gpt-oss-20b	Direct	1.75	51.8	52.7
gpt-oss-20b	Chain-of-Thought	1.53	50.4	51.1
gpt-oss-20b	Ledger	1.06	53.7	54.3
gpt-oss-20b	MUS-Repair	0.83	68.7	69.3
Qwen3-32B	Direct	0.00	28.9	28.9
Qwen3-32B	Chain-of-Thought	0.02	31.4	31.4
Qwen3-32B	Ledger	0.00	31.4	31.4
Qwen3-32B	MUS-Repair	0.00	38.2	38.2
Qwen3-8B	Direct	0.00	28.2	28.2
Qwen3-8B	Chain-of-Thought	0.00	27.9	27.9
Qwen3-8B	Ledger	0.00	25.2	25.2
Qwen3-8B	MUS-Repair	0.00	30.0	30.0

Table 10: Paired MUS-Repair tests against the strongest non-MUS comparator per model.

Model	Comparator	Δ Acc. (pp)	95% CI (pp)	p	q_{FDR}
gpt-oss-120b	Chain-of-Thought	8.01	[+6.19, +9.83]	<0.0001	<0.0001
gpt-oss-20b	Ledger	14.29	[+12.70, +15.93]	<0.0001	<0.0001
Qwen3-32B	Chain-of-Thought	6.49	[+4.34, +8.66]	<0.0001	<0.0001
Qwen3-8B	Direct	2.03	[+0.34, +3.76]	0.0178	0.0178

Table 11: Post-repair outcomes by trigger code in MUS traces.

Model	Trigger	Rows	Repair Acc. (%)	Repair SAT (%)
Qwen3-8B	Answer-Ledger Conflict	4131	4.0	100.0
Qwen3-8B	Incomplete Assignment	139	65.5	100.0
Qwen3-8B	Unsatisfiable Ledger	1	0.0	100.0
Qwen3-32B	Answer-Ledger Conflict	3643	5.5	99.6
Qwen3-32B	Incomplete Assignment	275	69.8	100.0
Qwen3-32B	Unsatisfiable Ledger	87	6.9	17.2
Qwen3-32B	Constraint Extraction Failure	2	0.0	100.0
gpt-oss-20b	Answer-Ledger Conflict	2402	33.3	100.0
gpt-oss-20b	Incomplete Assignment	735	33.3	99.9
gpt-oss-20b	Answer Parse Failure	449	23.4	100.0
gpt-oss-20b	Constraint Extraction Failure	429	22.6	100.0
gpt-oss-120b	Answer-Ledger Conflict	2471	21.2	99.9
gpt-oss-120b	Answer Parse Failure	416	18.5	99.8
gpt-oss-120b	Constraint Extraction Failure	32	12.5	96.9
gpt-oss-120b	Incomplete Assignment	30	26.7	100.0

Table 12: Pairwise overlap of MUS error rows across models.

Model A	Model B	Overlap	Jaccard	Share A	Share B
Qwen3-8B	Qwen3-32B	3143	0.726	0.792	0.897
Qwen3-8B	gpt-oss-20b	1687	0.416	0.425	0.950
Qwen3-8B	gpt-oss-120b	1945	0.470	0.490	0.919
Qwen3-32B	gpt-oss-20b	1638	0.450	0.467	0.923
Qwen3-32B	gpt-oss-120b	1892	0.507	0.540	0.894
gpt-oss-20b	gpt-oss-120b	1412	0.569	0.795	0.667

C Appendix Reproducibility Details

Generated tables and figures. All tables and figures in the paper are generated from the finalized evaluation artifacts with a deterministic build pipeline. Inferential statistics are computed from problem-level paired outcomes and then rendered into publication tables.

Runtime configuration. All reported runs use deterministic decoding with temperature set to zero. The execution policy fixes maximum repair attempts at two, maximum truncation retries at two, and ledger token budget at 3,000. For gpt-oss models, reasoning is left at the default configuration.

Claim registry. Quantitative statements in the paper are mapped to a claim registry and validated by an automated consistency pass before PDF generation.

Diagnostics snapshot. Qwen3-8B MUS repair attempts total 4,201, Qwen3-32B totals 3,858, gpt-oss-20b totals 2,598, and gpt-oss-120b totals 2,697. Trigger rates are 0.741 for Qwen3-8B, 0.680 for Qwen3-32B, 0.458 for gpt-oss-20b, and 0.475 for gpt-oss-120b. As of February 21, 2026, gpt-oss-120b has full coverage at 5,672 rows and 816 problems for DIRECT, CHAIN-OF-THOUGHT, LEDGER, and MUS-REPAIR.

D Example Transcripts

We present three annotated transcripts from Qwen3-8B evaluation runs, one per domain. Each transcript compares the DIRECT baseline (no constraint tracking) against MUS-REPAIR (ledger + Z3 verification + repair loop). Answers are shown as JSON assignment maps; ✓ denotes a correct (constraint-satisfying) solution and ✗ an incorrect one. These examples illustrate the failure modes discussed in Sections 5–6: state drift (the model silently violates earlier constraints), duplicate assignments, and stale repetition.

D.1 Transcript A: Scheduling (scheduling_249)

Setup. Six activities (Sync, Testing, Meeting, QA, Planning, Design) must be assigned start times and durations. Constraints accumulate over four turns.

Key observation. DIRECT carries forward incorrect default durations from turn 1 and never revises them, even after explicit duration constraints are introduced. MUS-REPAIR detects answer-ledger conflicts and produces valid assignments at every turn.

D.2 Transcript B: Logic Grid (logic_grid_021)

Setup. Four people (Blake, Drew, Avery, Finley) are each assigned a unique value in three categories: color (Red/Blue/Green/Yellow), pet (Cat/Dog/Bird/Fish), and profession (Doctor/Artist/Teacher/Chef).

Table 13: Scheduling transcript: DIRECT (1/4 correct) vs. MUS-REPAIR (4/4 correct).

Turn	New constraints	DIRECT answer		MUS-REPAIR answer	
1	QA must start between slots 1–2.	QA→1	✓	QA→2	✓
2	Testing ≠ Design (simult.); QA duration = 3; Design→slot 9.	QA dur = 3, Design→9, but Testing→5, Design→9 (Testing dur 2 ⇒ 5–6)	✗†	QA dur = 3, Design→9, Testing→4	✓
3	Testing duration = 3.	Testing dur = 3, Design→9 dur = 2, Meeting→9 dur = 2	✗	Testing→4 dur = 3; Design→9; QA→2 dur = 3	✓
4	Testing→slot 7; Planning→slot 5.	Testing→7 dur = 3, Design→9 dur = 2 (overlap 9–10)	✗	Testing→7 dur = 3, Planning→5, Design→9	✓

† DIRECT assigns default duration = 2 to several activities, violating the implicit single-slot default.

Table 14: Logic-grid transcript: DIRECT (0/5 correct) vs. MUS-REPAIR (5/5 correct).

Turn	New constraints	DIRECT answer		MUS-REPAIR answer	
1	Finley.pet < Drew.pet; Finley.pet ≠ Avery.pet.	Finley→Bird, Drew→Dog (Bird < Dog: wrong order)	✗	Finley→Cat, Drew→Dog	✓
2	Blake.color < Finley.color.	(identical to turn 1)	✗	Blake→Red, Finley→Yellow	✓
3	Drew.pet ≠ Finley.pet; Avery.pet ≠ Drew.pet; Avery.prof ≠ Finley.prof.	(identical to turn 1)	✗	Drew→Dog, Avery→Bird, Finley→Cat	✓
4	Drew → Chef.	Drew→Chef, Finley→Chef (**duplicate**: two Chefs)	✗	Drew→Chef, Finley→Artist	✓
5	Blake.color ≠ Drew.color; Drew ≠ Bird.	(same as turn 4, still two Chefs)	✗	Drew→Dog (not Bird); Blake→Red, Drew→Blue	✓

Key observation. DIRECT produces the same answer for turns 1–3 despite accumulating constraints, then introduces a duplicate profession assignment at turn 4 that it never corrects. MUS-REPAIR incrementally adjusts the assignment, maintaining uniqueness and ordering constraints throughout.

D.3 TRANSCRIPT C: SEATING (SEATING_062)

Setup. Seven people (Diana, Ruby, Tina, Noah, Charlie, Frank, Karen) sit around a round table with positions 1–7. Constraints include fixed positions, adjacency prohibitions, and separation requirements.

Table 15: Seating transcript: DIRECT (1/4 correct) vs. MUS-REPAIR (3/4 correct).

Turn	New constraints	DIRECT answer		MUS-REPAIR answer	
1	Karen→pos. 3; Karen not adjacent to Ruby.	Karen→3, Ruby→5	✓	Karen→3, Ruby→5	✓
2	Charlie not adjacent to Frank.	Charlie→6, Frank→7 (**adjacent**: violates new constraint)	✗	Charlie→7, Frank→4 (separated by 3 positions)	✓
3	Karen–Noah ≥1 apart; Tina–Frank ≥2 apart.	Frank→8 (**invalid**: only 7 seats)	✗	Noah→7, Tina→2, Frank→6	✓
4	Frank not adj. Ruby; Noah not adj. Charlie; Diana→pos. 6.	Karen→4 (**drift**: violates turn-1 at_position(Karen,3))	✗	Diana→6, but Ruby→5 adj. Frank→1 (wraps)	✗

Key observation. DIRECT exhibits three distinct failure modes across four turns: ignoring a new constraint (turn 2), producing an out-of-range position (turn 3), and drifting from a committed position (turn 4). MUS-REPAIR maintains consistency through turn 3 but fails at turn 4 when the constraint set becomes tightly coupled, illustrating the residual drift problem discussed in Section 6.