SOLVE, RECOVER, REUSE: LLM REASONING ACROSS WORKFLOWS WITH CORRUPTED CONTEXTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) increasingly operate in complex workflows and noisy contexts. It's important to have a comprehensive understanding of how model performance varies across workflow and corruption types, and whether both harmful and beneficial transitions arise under these challenges. We introduce an **Evolutionary Task Generator (ETG)** that generates a series of linked tasks across workflows and corruptions, providing a framework to systematically experiment and analyze: 1) model performance changes across workflows and corruptions, 2) the bidirectional transitions under corrupted contexts, and 3) training effects across workflows and corruptions. Across frontier LLMs, we find that model rankings and performance change substantially across workflows, e.g., DeepSeek-V4-Flash drops from 0.75 in direct solving to 0.20 in downstream reuse. Instance-level analysis shows that both harmful and beneficial transitions arise under corrupted contexts. Training results show that direct-solving gains may not transfer across workflows, and may even degrade model performance under complex workflows with corrupted contexts. Our results highlight the importance of examining LLMs reasoning through the lens of complex workflows and corrupted contexts, and suggest a more constructive direction when such noise is unavoidable in practice: resist the harmful and leverage the beneficial.

1 INTRODUCTION

As large language models (LLMs) move from isolated problem solving to complex workflows and noisy contexts, their ability to reason with imperfect interaction across workflows becomes increasingly important. Prior work has separately revealed the damage of corrupted context and complex workflows: irrelevant information in questions can distract models (Yang et al., 2025b); multi-turn interactions with underspecified instructions can degrade performance (Laban et al., 2026); noisy rationales can bias model's reasoning (Zhou et al., 2024).

Beyond studying models' performance in isolation, we need a comprehensive understanding of how model performance varies across workflow and corruption types. More importantly, beyond harmful transitions under corrupted contexts, we ask whether beneficial failure-to-success transitions also occur, which is preferred in practice: the model resists the harmful and leverages the beneficial. **Several important questions thus remain:** 1) How do LLMs performance vary across different workflows? 2) As frontier models become increasingly robust, which forms of corruption still damage their reasoning? 3) As corrupted context is typically studied for its harm, can beneficial transitions also occur under it? How to study the bidirectional effects? 4) Does improving a model's direct-solving ability lead to overall gain? And last but not least: how to construct data for the above controlled study?

To answer these questions, we introduce an **Evolutionary Task Generator (ETG)** that generates a series of linked tasks across three types of workflows and six types of corruptions. Our framework provides a better tool to systematically experiment, observe, and analyze: 1) model performance changes across workflows and corruptions, 2) the bidirectional transitions under corrupted contexts, and 3) training effects across workflows and corruptions. Starting from a seed problem, ETG progressively constructs tasks under three work modes (i.e., direct solving, explicit recovery, and downstream reuse), as shown in Fig. 1. Corrupted contexts are constructed based on two dimensions, i.e., Interaction Regime $\times$ Corruption Target, as shown in Fig. 2. ETG offers three key advantages:

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.

lineage preservation for full-chain monitoring of one instance across workflows and corruptions, **fine-grained reasoning structure annotations** for in-depth analysis, and **scalability** for large-scale evaluation and training without human annotation. Furthermore, we introduce two metrics to study bidirectional transitions under corrupted contexts, i.e., the beneficial failure-to-success transition α and the harmful success-to-failure transition β.

Using 5,636 evaluation tasks and 60,000 training tasks generated by ETG, we evaluate frontier LLMs and conduct controlled training experiments. Our experiments yield four main findings:

Model performance and model rankings change under different workflow modes. Strong direct-solving task performance does not ensure reliable recovery ability. A model can repair a solution when explicitly requested to do so, but it can still fail when given the same corrupted history with a simple downstream question. Most current benchmarks deal with direct-solving tasks, which might be unreliable for model selection in complex workflows (Cobbe et al., 2021). **Reasoning structure matters.** Despite LLMs robustness to surface-level context variation (Hua et al., 2025), their robustness remains limited when corruption dives into reasoning structure. For belief-conflict corruption, role relabeling for the two speakers produces large and directionally consistent performance shifts. **Both beneficial and harmful transitions occur under corrupted contexts.** Beyond the mainly studied harms caused by corrupted contexts, our analysis based on instance-level tracing reveals beneficial transitions under corrupted contexts. This suggests a more positive direction when noise is unavoidable in practice, that is to resist the harmful and leverage the beneficial.

Training on direct-solving tasks with clean contexts alone does not ensure improvement in complex workflows with corrupted contexts, and may even degrade model performance. The introduction of corrupted context data during training yields model performance improvements across workflows and corruptions. In particular, mixing corrupted data and clean direct-solving data yields more balanced improvements, with higher failure-to-success transition α and lower success-to-failure transitions β. Our experimental results suggest that in addition to the ability to solve the original task, models further need to learn how to resist the harmful, leverage the beneficial.

Our contributions are threefold:

- **Evolutionary Task Generator for full-chain monitoring.** We introduce ETG, a lineage-preserving, graph-annotated, and scalable task generator that evolves the same seed task across direct solving, explicit recovery, and downstream reuse under controlled corruptions.
- **Unified construction of workflows and corruptions.** We model different workflows and diverse corruptions along two dimensions, providing a systematic analysis of model performance changes across workflow modes and corruption types.
- **Bidirectional transitions under corrupted context during evaluation and training.** We study both the harmful and beneficial transitions under corrupted contexts through instance-level analysis during evaluation and training. Our findings suggest a more constructive direction when noise is unavoidable in practice, which is to *resist the harmful, and leverage the beneficial*.

2 Related works

With the wide application of LLMs (Vaswani et al., 2017), recent research has increasingly focused on complex interactive workflows (Laban et al., 2026) and in-context learning (Dou et al., 2026; Yao et al., 2022; 2023; Wei et al., 2022). Meanwhile, studies have separately revealed the challenges of corrupted context and complex workflows: irrelevant information in questions can distract models (Yang et al., 2025b); multi-turn interactions with underspecified instructions can degrade performance (Laban et al., 2026); noisy rationales can bias model's reasoning (Zhou et al., 2024). These studies typically examine specific workflow or corruption type. A systematic analysis therefore requires jointly considering multiple types of workflows and corruptions within a unified framework. However, many existing well-known datasets face the risk of data contamination (Yang et al., 2025c; He et al., 2026), and some cannot be automatically evolved to a series of tasks across workflows and corruptions (Cobbe et al., 2021; Mirzadeh et al., 2025), making them ill-suited for controlled studies of LLM reasoning across workflows and corrupted contexts (Ye et al., 2025a;b; Zheng et al., 2025; Ouyang et al., 2022; Rafailov et al., 2023; Shao et al., 2024).

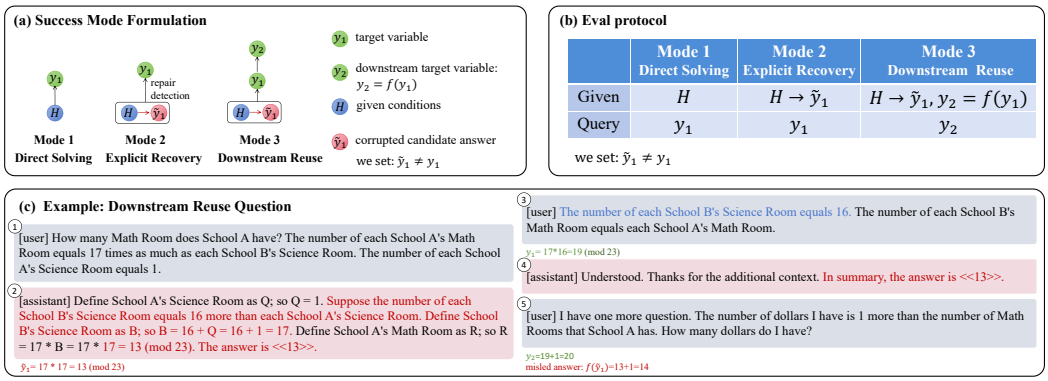

Figure 1: Three workflow modes built from the same seed task. (a) Task structures, (b) Evaluation protocols for direct solving, explicit recovery, and downstream reuse to infer $y_2 = f(y_1)$, and (c) An example of a downstream reuse question.

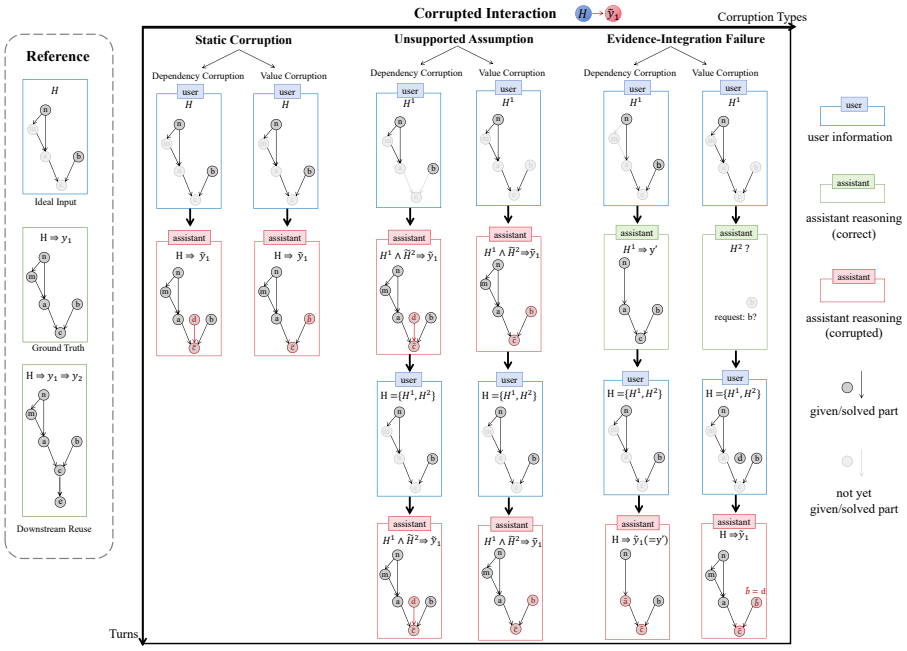

Figure 2: Controlled construction of six corruption types: Interaction Regimes × Corruption Targets. The left panel shows the reference task's input, groundtruth and downstream structure; the right panel shows interaction turns unfolding from top to bottom. Red highlights the corrupted structures.

3 Construction of Controlled Tasks

We introduce an **Evolutionary Task Generator (ETG)**. Starting from a seed problem with an annotated reasoning graph, ETG progressively constructs tasks across workflows under corruption contexts.

3.1 Three Modes: Direct Solving, Explicit Recovery and Downstream Reuse

Rather than independently constructing clean, corrupted, and downstream tasks, we progressively evolve each task from the same latent reasoning graph, enabling full-chain monitoring.

We design three working modes to examine agent reasoning under different roles. Given a problem in which the conditions H entail a target answer y_1, Fig. 1 illustrates the three evaluation protocols.

Table 1: Our corrupted-context generation mechanisms: 2 corruption targets (*Value* vs. *Dependency*) $\times$ 3 interaction regimes.

	Generation Mechanism: Targets $\times$ Regimes		Interaction Properties		
Corruption	**Corruption Target**	**Interaction Regime**	**Fully Specified**	**Turns**	**Error Progression**
Static-Value Corruption	Value	Static Corruption	✓	Single	Fully corrupted
Static-Dependency Corruption	Dependency	Static Corruption	✓	Single	Fully corrupted
Assumption-Value Corruption	Value	Unsupported Assumption	×	Multi	Fully corrupted
Assumption-Dependency Corruption	Dependency	Unsupported Assumption	×	Multi	Fully corrupted
Integration-Value Corruption	Value	Evidence-Integration Failure	×	Multi	Initially valid $\rightarrow$ later corrupted
Integration-Dependency Corruption	Dependency	Evidence-Integration Failure	×	Multi	Initially valid $\rightarrow$ later corrupted

Mode 1: Direct Solving. The model is given the correct conditions H and is asked to derive y_1.

Mode 2: Explicit Recovery. The model is given an interaction context containing H with a reasoning process that produces a corrupted candidate answer $\tilde{y}_1 \neq y_1$. It is asked to first determine whether the context is erroneous and, if so, to recover the solution. The corrupted interaction contains deliberately constructed structural reasoning errors, which we describe in the next section.

Mode 3: Downstream Reuse. The model is given the same correct conditions H and the corrupted interaction leading to $\tilde{y}_1$, together with a downstream target y_2 defined by a simple linear relation $y_2 = f(y_1)$. Instead of directly asking the model to repair y_1, we ask it to solve for y_2. This setting tests whether the model can implicitly recover the information required to infer the correct y_1 and use it in a new downstream problem, rather than being misled by the erroneous $\tilde{y}_1$ already present in the context.

Importantly, the three modes are not constructed as independent tasks; instead, we progressively transform each instance across three working modes. Mode 2 is obtained by injecting a structural reasoning corruption into the corresponding Mode 1 instance, and the corrupted $\tilde{y}_1$ interaction used in Mode 3 is directly inherited from Mode 2. This progressive construction enables instance-level tracing of the model's behavior throughout the entire working pipeline. Figure 1 summarizes the success criterion for each mode.

3.2 Corruption Taxonomy: Interaction Regimes $\times$ Corruption Targets

Our corruptions focus on reasoning structures rather than surface-level linguistic perturbations. We construct corruptions from two dimensions: **Interaction Regimes** $\times$ **Corruption Targets**. The corruption taxonomy is summarized in Table 1 and Fig. 2.

Before corruption, we use a logic generator to synthesize annotated tasks (Ye et al., 2025a). Each task is associated with a ground-truth topology graph, which records the values of all variables involved in the reasoning process as nodes, and their dependency relations as edges.

Based on the ground-truth reasoning structure, we introduce corruptions to simulate complex and non-ideal reasoning processes. We consider two types of corruption targets: *value*, the value of a variable represented by a node in the topology graph; *dependency*, the dependency relation between variables represented by an edge. We construct errors under three interaction mechanisms:

A. Static Corruption. The problem is fully specified before the corrupted candidate is introduced. This represents the most idealized interaction setting: all valid conditions are provided at once, while the candidate solution contains an error.

B. Unsupported Assumption. In a multi-turn interaction with incomplete information, the user does not provide all conditions at once. For a missing condition H^2, the model makes an unsupported assumption $\tilde{H}^2$ at an early stage. The user later provides the correct information H^2, but the model fails to update its reasoning and continues to rely on its earlier incorrect assumption.

C. Evidence-Integration Failure. In a multi-turn interaction with incomplete information, the model recognizes the missing information and either queries the missing variable or computes an intermediate variable from the available evidence. When new evidence arrives, however, the model fails to correctly integrate it into the task state (i.e., substituting the new variable incorrectly, or failing to incorporate a newly established dependency).

3.3 A DEEPER VIEW: BIDIRECTIONAL TRANSITIONS UNDER CORRUPTED CONTEXT

Existing studies on corrupted context have revealed its damaging effects (Laban et al., 2026; Yang et al., 2025b; Zhou et al., 2024). Yet context is introduced precisely to expand the task space and enable models to leverage additional information, while corruption is often unavoidable in practice. We therefore ask: **does corrupted contexts only lead to performance degradation?**

Our instance-level task evolution method enables full-chain monitoring of the same instances across the workflow. For the transition from $Mode_i$ to $Mode_{i+1}$, where $i \in \{1, 2\}$, we define two transition metrics to characterize the bidirectional transition under corrupted context:

$$\alpha_i = P(M_{i+1} = 1 \mid M_i = 0), \qquad \beta_i = P(M_{i+1} = 0 \mid M_i = 1), \tag{1}$$

where $M_i = 1$ indicates that an instance is solved correctly in Mode-i, and $M_i = 0$ otherwise. Here, beneficial α and harmful β describe observed transitions across workflows, not causal effects of corruption itself.

β_1 captures the failure pattern most commonly studied in prior work: the model can solve an instance directly, but fails to recover the correct answer under corrupted contexts. In contrast, α_1 captures a desirable behavior in practice: when the model cannot solve the problem directly, an additional reasoning context, despite containing errors, can help. β_2 measures whether successful recovery remains reliable during downstream reuse: the model can recover the correct state in Mode-2, but is also misled by the corrupted context when its attention shifts to the downstream task. Furthermore, we analyze *how* downstream failures occur by checking whether wrong $\tilde{y}_2$ follows from the corrupted upstream state: $\tilde{y}_2 = f(\tilde{y}_1)$. Conversely, α_2 captures cases where explicit recovery fails in Mode-2, yet the model still answers the downstream task correctly in Mode-3.

4 EXPERIMENTS AND RESULTS

Experimental Setup. We control the reasoning depth from 1 to 8 steps, with seed tasks uniformly distributed across these eight levels. We construct **ETG-Eval** with 5,636 evaluation tasks: 200 direct-solving seed tasks are evolved into 2,765 explicit-recovery tasks, which are further evolved into 2,671 downstream-reuse tasks. We also construct **ETG-Train** with 60,000 training tasks, consisting of 30,000 direct-solving tasks and 30,000 corresponding explicit-recovery tasks. Downstream-reuse tasks are excluded from training to evaluate cross-workflow generalization. For efficiency during training, we evaluate on **ETG-Eval-Small**, a 1,153 ETG-Eval subset sampled approximately evenly from the three modes. We evaluate frontier LLMs: Claude-Opus-5, Gemini-3.6-Flash, DeepSeek-V4-Flash, GPT-5.5, GPT-5.4, and GPT-4o. Because some cases involve many reasoning steps, **we perform all computations modulo 23** to prevent numerical values from growing excessively large.

4.1 PERFORMANCE VARIES AND RANKINGS CHANGE

Table 2 reports frontier-model performance on the same tasks under different workflow modes and corruption structures (Hurst et al., 2024; Xu et al., 2026).

First, model rankings change across modes. We order models by their Mode-1 direct-solving performance, which corresponds to common benchmarks. However, this ordering does not persist once the models are evaluated under recovery or downstream-reuse settings. Notably, Claude-Opus-5, the strongest model in Mode-1, exhibits the highest misled rate in Mode-3, even though all downstream instances are derived from the Mode-1 seed tasks.

Second, performance within a single model can shift sharply. For example, DeepSeek-V4-Flash drops from 74.5% accuracy in Mode-1 to 20.22% in Mode-3. Thus, being a strong direct solver does not imply strong explicit-recovery ability, and successful explicit recovery does not necessarily prevent the model from being misled during downstream reuse. Robustness also varies across corruption types: for GPT-5.5, Mode-2 recovery accuracy ranges from 40.27% to 89.69% across the corruption types.

Table 2: Performance across three workflow modes. Mode 1 reports direct-solving accuracy. Mode 2 reports recovery accuracy (Acc.) and error-detection rate (D-Rate.) under three interaction regimes. Mode 3 reports downstream accuracy and the misled rate among failures (**Misled**, lower is better). Cell colors are rank-based within each column (red is better, blue is worse).

Model	**Mode 1**	**Mode 2: Explicit Recovery**						**Mode 3**	
	Direct Solving	**Static**		**Assumption**		**Integration**		**Downstream Reuse**	
	Acc. ↑	Acc.	D-Rate.	Acc.	D-Rate.	Acc.	D-Rate.	**Acc.** ↑	**Misled** ↓
GPT-4o	0.490	0.470	0.642	0.103	0.507	0.183	0.543	0.064	0.978
GPT-5.4	0.590	0.604	0.981	0.412	0.949	0.474	0.872	0.173	0.952
DeepSeek-V4-Flash	0.745	0.784	0.994	0.375	0.987	0.652	0.915	0.202	0.957
Gemini-3.6-Flash	0.820	0.839	**1.000**	0.456	**0.999**	0.869	**0.986**	0.596	0.816
GPT-5.5	0.890	0.897	**1.000**	0.403	0.996	**0.872**	0.965	**0.732**	**0.755**
Claude-Opus-5	**0.980**	**0.905**	0.990	**0.672**	0.996	0.844	0.971	0.582	0.986

I: Direct-solving benchmarks alone may be insufficient for model selection under complex workflows. Model rankings can reverse and performance can vary across workflow modes.

4.2 Reasoning Structure Matters

While recent frontier LLMs can be relatively robust to surface variations in prompts(Hua et al., 2025), Table 2 shows their robustness remains limited when corruption dives into reasoning structure. We further perform a controlled role ablation. Specifically, we replace the `user-assistant` labels in the interaction used in Table 2 with the labels `user1-user2`, other content unchanged.

Table 3 shows that role relabeling has a larger and directionally consistent effect on unsupported-assumption errors, but a smaller and less consistent effect on evidence-integration failures.

For unsupported assumptions, role relabeling changes the attribution of the conflicting belief, influencing reasoning structure. The lower performance of User-Assistant suggests models' stronger anchoring on assistant's false belief (model itself's false belif). In contrast, for evidence-integration failures, role labels are superficial because the error is role-independent.

Table 3: **Role label ablation.** Recovery accuracy of the models

Model	**Role**	**Assumption**	**Integration**
Gemini-3.6	U–A	0.4555	**0.8685**
	U1–U2	**0.8495**	0.8529
GPT-5.4	U–A	0.4115	0.4740
	U1–U2	**0.4888**	**0.4862**
GPT-4o	U–A	0.1026	0.1834
	U1–U2	**0.6940**	**0.2543**

II: Robustness degrades once corruptions hit the reasoning structure. For corruption of belief conflicting from two speakers, role relabeling produces large and directionally consistent performance shifts.

4.3 The Bidirectional Transitions Under Corrupted Context

Despite existing work emphasizing success-to-failure transitions under corrupted contexts (Laban et al., 2026; Yang et al., 2025b; Zhou et al., 2024), our instance-level tracing shows that the reverse transition also occurs: some instances change from failure to success.

As shown in Fig. 3 and Table 4, we quantify these bidirectional effects using α and β. Consistent with prior work, corrupted context can cause damage: for GPT-4o, β_1 can reach 0.897 and β_2 0.990, and Claude-Opus-5 reaches 0.326 and 0.505, respectively. However, beneficial transitions under corrupted context are also non-negligible. For example, Claude-Opus-5 achieves α as high as 0.633, exceeding its corresponding β in the same setting. Such beneficial transitions are particularly desirable in practice and therefore need to be explicitly identified and analyzed. The balance between beneficial and harmful transitions varies substantially across workflow modes and corruption structures. On average, the recovery transition exhibits higher α and lower β than downstream reuse,

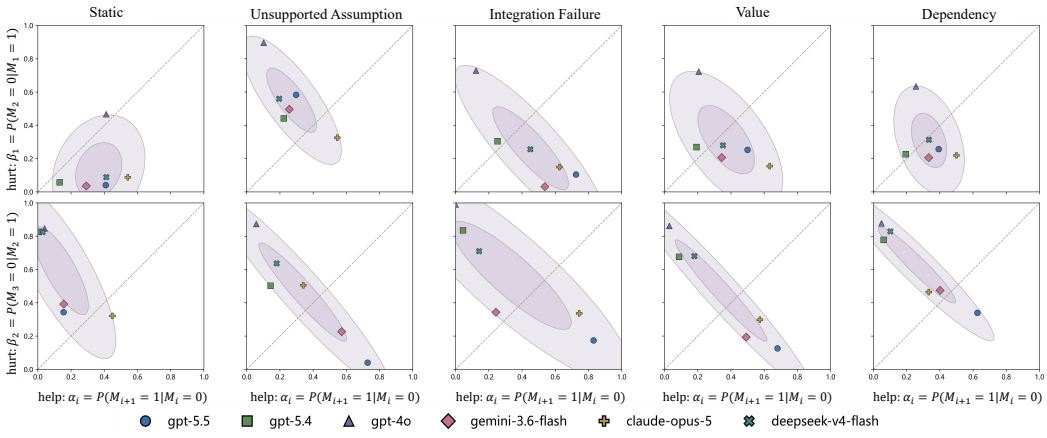

Figure 3: **Bidirectional transition profiles across corruption categories.** Each point represents a model in the (α_i, β_i) plane. The top and bottom rows correspond to direct-solving to explicit-recovery (i=1) and explicit-recovery to downstream-reuse (i=2). Columns show five corruption categories. Ellipses are centered at the empirical mean and oriented by the covariance; their inner and outer semi-axes span one and two standard deviations along the principal directions, respectively. The dashed diagonal serves as a visual reference.

Table 4: Bidirectional transition probabilities across models and corruption categories. Each cell reports α_i/β_i. Panel (a) corresponds to explicit recovery ($i = 1$), and Panel (b) to downstream reuse ($i = 2$).

(a) Explicit Recovery Transition ($i = 1$)**:** α_1/β_1					
Model	**Static**	**Assumption**	**Integration**	**Value**	**Dependency**
GPT-5.4	0.132 / 0.058	0.223 / 0.442	0.253 / 0.304	0.193 / 0.269	0.195 / 0.227
GPT-4o	0.412 / 0.467	0.102 / 0.897	0.123 / 0.729	0.206 / 0.722	0.256 / 0.634
Gemini-3.6-Flash	0.292 / 0.036	0.256 / 0.496	0.537 / 0.032	0.345 / 0.207	0.333 / 0.206
Claude-Opus-5	0.542 / 0.088	0.545 / 0.326	0.625 / 0.149	0.633 / 0.155	0.500 / 0.220
GPT-5.5	0.409 / 0.041	0.298 / 0.583	0.725 / 0.104	0.500 / 0.253	0.393 / 0.257
DeepSeek-V4-Flash	0.412 / 0.089	0.196 / 0.560	0.450 / 0.256	0.352 / 0.280	0.334 / 0.313
(b) Downstream Reuse Transition ($i = 2$)**:** α_2/β_2					
Model	**Static**	**Assumption**	**Integration**	**Value**	**Dependency**
GPT-5.4	0.004 / 0.826	0.143 / 0.502	0.044 / 0.835	0.088 / 0.676	0.062 / 0.778
GPT-4o	0.040 / 0.847	0.058 / 0.874	0.000 / 0.990	0.029 / 0.862	0.049 / 0.878
Gemini-3.6-Flash	0.156 / 0.392	0.572 / 0.227	0.243 / 0.343	0.492 / 0.194	0.401 / 0.474
Claude-Opus-5	0.450 / 0.321	0.340 / 0.505	0.744 / 0.337	0.574 / 0.299	0.333 / 0.465
GPT-5.5	0.155 / 0.343	0.728 / 0.040	0.831 / 0.174	0.680 / 0.126	0.626 / 0.339
DeepSeek-V4-Flash	0.029 / 0.825	0.180 / 0.637	0.141 / 0.710	0.180 / 0.681	0.102 / 0.829

suggesting that models can use corrupted context during explicit recovery, but may still be misled by it in downstream tasks. Within recovery, unsupported-assumption contexts are less likely to help and more likely to hurt than the other corruption regimes.

These results demonstrate the bidirectional transitions under corrupted context and reveal a beneficial transition that has received comparatively less attention. More broadly, an operation should not be dismissed solely because it decreases aggregate accuracy; instead, its beneficial and harmful transitions should be analyzed separately to understand how its useful effects can be preserved while its failures are mitigated.

III: Although the harm of corrupted contexts is mainly studied, our instance-level tracing reveals substantial beneficial transitions. This suggests a more positive direction when noise is unavoidable in practice: resist the harmful, leverage the beneficial.

4.4 TRAINING WITH DIFFERENT STRATEGIES

The former results reveal substantial performance variation across workflow modes. We delve further into the performance during training. In particular, **we ask: does improving a model's direct-solving ability necessarily benefit its performance across different workflows?**

To investigate this question, we construct three training strategies that differ in the reasoning context presented during training. **Clean** data follow the Mode-1 format in Fig. 1(b): given H, the model produces y_1 together with its reasoning process. **Corrupted** data follow the Mode-2 format: the input contains $H \rightarrow \tilde{y}_1$, i.e., an interaction history with corrupted reasoning, and the model is trained to recover y_1 with a corrected reasoning process. **Mixed** data are constructed by randomly sampling half of the clean and half of the corrupted examples and combining them in a 1:1 ratio.

We perform full-parameter fine-tuning on *Llama-3.2-3B-Instruct* and *Qwen3-4B-Instruct* (Yang et al., 2025a). Table 5 shows that improving direct-solving ability does not necessarily benefit performance across different workflows: the gains can be marginal and may even induce performance decrease. For Qwen3-4B, training only on clean Mode-1 data improves direct-solving accuracy to 0.700, but reduces accuracy across Mode-2 and Mode-3 compared with the base model, with the largest drop reaching 18.9%. For Llama-3B, clean training yields only modest improvements of roughly 2-5% in Mode-2 and Mode-3.

We further analyze how training changes the bidirectional transition probabilities α and β. As shown in Fig 4, increasing α while decreasing β is the desirable direction, corresponding to $(+1, -1)$ in the α-β plane: resist the harmful, leverage the beneficial. Conversely, $(-1, +1)$ indicates a deterioration in both aspects. Consistent with our preceding results, clean-only training has the risk of bringing higher β and lower α, which is undesirable.

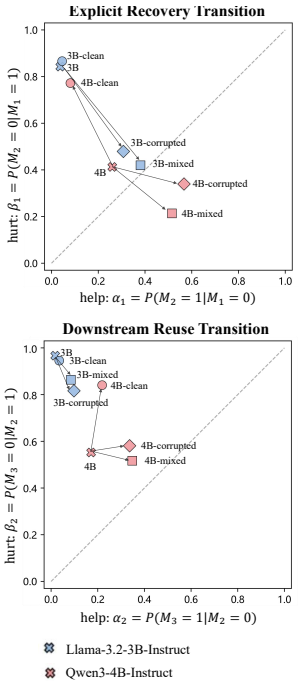

Figure 4: Bidirectional effects under different training strategies.

In contrast, once corrupted-context recovery examples are included during training, improvements extend beyond the training workflow. As shown in Table 5, both *Corrupted* and *Mixed* training improve recovery and can also benefit downstream reuse and direct solving. However, training solely on corrupted data maybe aggressive and less stable. For example, the Llama-3B model trained on corrupted data reaches only 0.225 accuracy in Mode-1. Mixed training provides a more balanced improvement across workflows: although it is not always the best, its performance is generally close to top-1. As shown in Fig 4, mixed training more consistently moves models toward higher α and lower β, suggesting an improved ability to exploit informative context while resisting its corruption. Moreover, the downstream-reuse points are more concentrated in the low-α, high-β region than those for explicit recovery, further indicating that reliable downstream reuse poses a greater challenge than explicit recovery.

We further examine whether this corruption-awareness generalizes across workflows. As our training data contain only Mode-1 and Mode-2 examples, Mode-3 provides a direct test of cross-workflow generalization. Table 5 shows that training with corrupted data improves Mode-3 accuracy while reducing the misled rate, with both changes exceeding 10%. However, Mode-3 performance remains relatively low compared with the stronger Mode-1 and Mode-2 results, suggesting considerable space for improvement as LLMs are deployed in more complex workflows.

Training results suggest that, as LLMs move from idealized problem solving to complex workflows, improving Mode-1 reasoning alone is insufficient and may even degrade performance in other workflows. Models must jointly acquire the ability to solve the original task, and be equipped with corruption-awareness: resist the harmful, leverage the beneficial.

IV: As LLMs move from idealized problem solving to complex workflows, improving direct-solving ability alone is insufficient and **may even degrade performance** in other workflows.

Table 5: Performance under different training strategies. Mode-1 reports direct-solving accuracy. For each Mode-2 corruption regime, entries report recovery accuracy with error-detection rate in parentheses. Mode-3 reports downstream accuracy with the misled rate among failures in parentheses. No Mode-3 examples are used during training.

Model	Training	Mode 1: Direct Solving	Mode 2: Explicit Recovery			Mode 3: Downstream Reuse
		Acc.	Static	Assumption	Integration	Acc. (Misled)
Llama-3.2-3B	no-training	0.200	0.1368 (0.9273)	0.0000 (0.0000)	0.0000 (0.0000)	0.0185 (0.9887)
	Clean	0.675	0.1795 (0.5299)	0.0485 (0.0874)	0.0382 (0.3359)	0.0369 (0.9368)
	Mixed	**0.700**	**0.5940** (**1.0000**)	**0.5874** (**0.8301**)	0.2443 (0.6565)	0.1107 (0.8423)
	Corrupted	0.225	0.5684 (0.8803)	0.0687 (0.1758)	**0.2621** (0.4126)	**0.1273** (**0.6722**)
Qwen3-4B	no-training	0.125	0.4701 (0.9786)	0.1942 (0.7621)	0.1450 (0.8015)	0.2546 (0.8776)
	Clean	**0.700**	0.3974 (0.5940)	0.0049 (0.0049)	0.0534 (0.2137)	0.2085 (0.6729)
	Mixed	0.600	0.7137 (**1.0000**)	**0.7233** (**0.9951**)	**0.4733** (**0.9771**)	**0.4373** (**0.3887**)
	Corrupted	0.600	**0.7650** (**1.0000**)	0.5583 (0.9903)	0.4580 (**0.9771**)	0.3875 (0.4316)

5 Discussion and Limitations

Reasoning-structure Corruption. Our study focuses on corruptions that alter the reasoning structure rather than surface-level linguistic variations. As LLMs move from idealized problem-solving settings to flexible and noisy workflows, they face new challenges such as incomplete information, conflicting beliefs among speakers, and difficulties in updating the task state as new evidence arrives. Although these challenges may appear to be merely surface-level changes in the interaction context, they fundamentally alter the reasoning structure These structural changes are the primary focus of our study.

Controlled Study We conduct a controlled study using a generator that produces scalable, lineage-preserving, and fine-grained annotated tasks across workflows and corruptions. Our goal is not to reproduce a specific real-world application, but to obtain verifiable and quantitatively reliable conclusions. In open real-time interactions, conversational formats and reasoning trajectories change dynamically, introducing additional sources of evaluation noise. When reproducing some real-time interaction work's experiments, we observed a non-negligible rate of evaluator judgment errors and distortions of the intended input information. These issues do not undermine their qualitative findings, but they make reliable quantitative analysis difficult (Laban et al., 2026). Controlled studies and application-specific evaluations therefore serve different primary purposes. Controlled studies use scalable, lineage-preserving, and annotated data to derive scientifically grounded conclusions and insights with broader guidance value (Ye et al., 2025a). Application-specific evaluations aim to identify problems in particular environments, while drawing guidance from controlled studies.

Interpreting bidirectional transition Our transition metrics are descriptive rather than causal. Our goal is not to assess whether corrupted context is more beneficial than its clean counterpart; rather, we ask whether models can still exploit useful information in the context when errors are unavoidably present, instead of being uniformly harmed by the corruption. A failure-to-success transition under corrupted context does not imply that the corruption itself causes the improvement. We study whether models can still effectively leverage additional context despite the presence of corruption.

6 Conclusion

We study LLM reasoning beyond isolated direct solving by examining how model behavior changes across workflow modes and corruption types. We introduce ETG, a lineage-preserving and graph-annotated framework that evolves the same seed task across direct solving, explicit recovery, and downstream reuse under six controlled corruption types. Across frontier LLMs, we find substantial shifts in both performance and model rankings across workflows, while reasoning-structure corruptions remain challenging even for strong models. Instance-level transition analysis further reveals that both harmful and beneficial transitions occur under corrupted contexts. Training experiments show that improving direct solving alone does not necessarily transfer across workflows and may even degrade performance, whereas incorporating corrupted-context examples can improve recovery and transfer to unseen downstream reuse. Together, these findings motivate evaluating and training LLM reasoning across workflows and corruptions.

AI USE STATEMENT

In this work, we used generative AI tools for [assist with translation, provide feedback on research methodology or experiments, generate synthetic datasets only for the exploratory auto-research analysis experiment reported in the appendix]. We have not used generative AI tools for [Generate synthetic data sets for our studies, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design research methodology or experiments, implement methods, clean and reformat dataset, support qualitative and thematic data analysis, interpret results], and [assist in the writing of proofs] are not applicable to this work.

We did not use generative AI tools to generate the synthetic datasets for the main ETG evaluation and training experiments. In the exploratory auto-research experiment in Appendix, we used a generative AI coding agent to design and implement a perturbation generator, which produced a separate synthetic training dataset.

We have reviewed all AI-assisted work. [LLM-generated content was verified by 2 authors]. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

not be more than 1 page.

REFERENCES

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.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. Cl-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026.

Xinyi He, Ying Yang, Chuanjian Fu, Sihan Guo, Songchun Zhu, Lifeng Fan, Zhenliang Zhang, and Yujia Peng. Automatic cognitive task generation for in-situ evaluation of embodied agents. *arXiv preprint arXiv:2602.05249*, 2026.

Andong Hua, Kenan Tang, Chenhe Gu, Jindong Gu, Eric Wong, and Yao Qin. Flaw or artifact? rethinking prompt sensitivity in evaluating llms. In *Proceedings of the 2025 conference on empirical methods in natural language processing*, pp. 19900–19910, 2025.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *International Conference on Learning Representations*, volume 2026, pp. 54738–54778, 2026.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, volume 2025, pp. 94743–94765, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Yang Wang, and Liangming Pan. How is llm reasoning distracted by irrelevant context? an analysis using a controlled benchmark. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13340–13358, 2025b.

Yue Yang, Shuibo Zhang, Kaipeng Zhang, Yi Bin, Yu Wang, Ping Luo, and Wenqi Shao. Dynamic multimodal evaluation with flexible complexity by vision-language bootstrapping. In *International Conference on Learning Representations*, volume 2025, pp. 21399–21420, 2025c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. In *International Conference on Learning Representations*, volume 2025, pp. 97699–97709, 2025a.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.2, how to learn from mistakes on grade-school math problems. In *International Conference on Learning Representations*, volume 2025, pp. 78136–78147, 2025b.

Xinyue Zheng, Haowei Lin, Shaofei Cai, Yaodong Yang, Zilong Zheng, and Yitao Liang. Unicode: Augmenting evaluation for code reasoning. *arXiv preprint arXiv:2510.17868*, 2025.

Zhanke Zhou, Rong Tao, Jianing Zhu, Yiwen Luo, Zengmao Wang, and Bo Han. Can language models perform robust reasoning in chain-of-thought prompting with noisy rationales? *Advances in Neural Information Processing Systems*, 37:123846–123910, 2024.

Table 6: Sensitivity to surface-form rephrasing. The rephrased version changes only the wording style while preserving task content and reasoning structure.

		Mode 1	Static Corruption		Evidence-Integration Failure	
Model	**Wording**	Acc.	Acc.	D-Rate.	Acc.	D-Rate.
GPT-4o	Original	0.4900	0.4699	0.6418	0.1834	0.5433
	Rephrased	0.4350	0.5473	0.7045	0.1799	0.5294
GPT-5.4	Original	0.5900	0.6040	0.9811	0.4740	0.8720
	Rephrased	0.5800	0.5636	0.9905	0.4844	0.8685

A Appendix

A.1 Evaluation Prompts

All evaluation settings use the same system prompt, which specifies the task domain, arithmetic rules, default values for undetermined quantities, and the required answer format.

Shared system prompt

```
You're an expert at solving elementary math problems involving
    addition, subtraction, and multiplication. You solve all the
    problems in a uniform format. Our scenarios involve up to four
    categories of objects: schools, classrooms, backpacks and
    stationeries. Each school may contain classrooms, each classroom
    may contain backpacks, and each backpack may contain stationeries.
     All calculations are done modulo 23. For example, 3 + 2 equals 5,
     1 + 22 equals 0, and 22 + 2 equals 1. Any quantity that cannot be
     determined, either directly or indirectly, from the stated
    information defaults to 0. When providing your solution, please
    end with 'The final answer is <<x>>.' where x is your final answer
    .
```

A.2 Additional Results

A.2.1 Sensitivity to Surface-Form Rephrasing

To examine whether the main observations depend on a particular wording style, we construct a surface-form variant that rephrases the task descriptions while preserving the underlying task content, reasoning graph, corruption structure, and evaluation protocol. Table 6 compares the original and rephrased versions for GPT-4o and GPT-5.4. We report direct-solving accuracy in Mode 1 and recovery accuracy (Acc.) together with error-detection rate (D-Rate.) for two representative corruption regimes.

The surface-form changes produce both increases and decreases, with no consistent directional shift. The main patterns are not tied to the particular wording used in the primary evaluation.

A.2.2 Confidence Intervals for Workflow Accuracy and Repeated results

To quantify uncertainty arising from the sampled task instances, we report 95% percentile bootstrap confidence intervals for the accuracy results in the main experiments. We perform 10,000 bootstrap resamples at the seed-problem level. Whenever a seed problem is sampled, all of its lineage-derived instances are retained together, thereby preserving the dependence among the corresponding Solve, Recover, and Reuse examples. Each metric is recomputed from its pooled numerator and denominator in every bootstrap replicate.

Table 7: Accuracy with 95% seed-level cluster-bootstrap confidence intervals. Recover accuracy is end-to-end: an example is counted as correct only when the model identifies the error and produces the correct answer. Values are reported as estimate [95% CI].

Model	Solve	Recover: Static	Recover: Assumption	Recover: Integration	Reuse
GPT-5.5	.890 [.845, .930]	.897 [.859, .932]	.403 [.364, .444]	.872 [.840, .902]	.732 [.704, .759]
GPT-5.4	.590 [.520, .655]	.604 [.541, .667]	.412 [.366, .458]	.474 [.415, .534]	.173 [.145, .204]
GPT-4o	.490 [.420, .560]	.470 [.433, .509]	.103 [.084, .123]	.183 [.150, .219]	.064 [.041, .090]
Gemini-3.6-Flash	.820 [.765, .870]	.839 [.790, .887]	.456 [.413, .499]	.869 [.822, .910]	.596 [.563, .630]
Claude-Opus-5	.980 [.960, .995]	.905 [.876, .930]	.672 [.634, .709]	.844 [.807, .880]	.582 [.558, .605]
DeepSeek-v4-Flash	.745 [.685, .805]	.784 [.738, .828]	.375 [.338, .414]	.652 [.600, .702]	.202 [.178, .227]

Table 8: Repeat evaluations using the same test instances. Recover columns report end-to-end correction accuracy/detection rate. Reuse reports accuracy/misled rate among incorrect responses.

Model	Run	Solve	Recover: Static	Recover: Assumption	Recover: Integration	Reuse
GPT-5.4	R1	.590	.604/.981	.412/.949	.474/.872	.173/.952
GPT-5.4	R2	.580	.612/.979	.416/.958	.486/.888	.180/.943
GPT-4o	R1	.490	.470/.642	.103/.507	.183/.543	.064/.978
GPT-4o	R2	.525	.495/.659	.087/.487	.170/.528	.061/.982
GPT-4o	R3	.485	.476/.649	.079/.463	.190/.531	.060/.981

The confidence intervals preserve the principal patterns reported in the main text. Unsupported-assumption corruption yields the lowest Recover accuracy for all six models. High Solve accuracy also does not imply comparably strong Reuse performance: Claude-Opus-5 achieves the highest Solve accuracy, whereas GPT-5.5 achieves the highest Reuse accuracy. Consequently, model rankings depend on the workflow rather than following a single direct-solving hierarchy. These intervals quantify uncertainty over sampled task lineages; their purpose is estimation rather than exhaustive pairwise significance testing.

We repeated the complete evaluation for GPT-5.4 and GPT-4o using exactly the same test instances and ground-truth answers. GPT-5.4 was evaluated twice and GPT-4o three times. Across the reported aggregate metrics, the mean per-metric range was 0.009 for GPT-5.4 and 0.020 for GPT-4o, respectively. More importantly, every evaluation preserved the same structural findings: recovery from static corruption was consistently stronger than recovery from integration failures and unsupported assumptions; detection rate remained higher than end-to-end correction accuracy; and downstream Reuse accuracy remained lower than direct Solve accuracy. The aggregate results and the qualitative conclusions derived from them are stable across repeated evaluations.

A.2.3 Exploratory RL Experiments

Beyond supervised fine-tuning, we explore whether the same framework can characterize models trained with preference- and reward-based objectives. We construct 30,000 training prompts from the same corrupted-interaction data used for supervised training. For DPO, the ground-truth solution is used as the chosen response and the corrupted solution as the rejected response. For GRPO, a generated answer receives a reward of $+1$ when it matches the ground-truth answer, -1 when it matches the corrupted answer or cannot be parsed, and -0.5 when it is another valid value in $\{0, \ldots, 22\}$. These experiments contain no downstream-reuse training examples.

We evaluate DPO when initialized either directly from the base model or from the corresponding mixed-SFT checkpoint. GRPO is evaluated from the mixed-SFT checkpoint. Table 9 reports the results under the same three-workflow evaluation protocol used in the main experiments.

The clearest pattern is the importance of initialization. For both model families, applying DPO after mixed SFT substantially outperforms applying DPO directly to the base model across the evaluated workflows.

Table 9: Exploratory DPO and GRPO results. Mode-2 cells report recovery accuracy and error-detection rate as Acc./D-Rate. Mode 3 reports downstream accuracy and the misled rate among failures as Acc./Misled, where lower Misled is better.

Model	Objective	Initialization	Mode 1	Static	Assumption	Integration	Mode 3
Llama-3.2-3B	DPO	Base	0.225	0.1624/0.9188	0.0000/0.0000	0.0000/0.0000	0.0203/0.9631
Llama-3.2-3B	DPO	Mixed SFT	0.700	0.6496/1.0000	0.6699/0.9078	0.2901/0.7710	0.1162/0.8267
Llama-3.2-3B	GRPO	Mixed SFT	0.700	0.6282/1.0000	0.6068/0.8495	0.2595/0.6870	0.1125/0.8462
Qwen3-4B	DPO	Base	0.150	0.5043/0.9701	0.1845/0.7621	0.1603/0.8092	0.2601/0.8445
Qwen3-4B	DPO	Mixed SFT	0.575	0.6966/1.0000	0.7621/0.9951	0.4809/0.9771	0.4410/0.3910
Qwen3-4B	GRPO	Mixed SFT	0.600	0.7436/1.0000	0.7379/0.9951	0.4656/0.9771	0.4539/0.3922

Table 10: Exploratory evaluation of autonomously designed perturbation data. Recover columns report end-to-end correction accuracy/detection rate. Reuse reports accuracy/misled rate among incorrect responses. The Solve comparison is directly controlled under the matched model and training setting; Recover and Reuse are reported as diagnostic transfer evaluations because the autonomously selected perturbations need not match our workflow-oriented taxonomy.

Training data	Solve	Recover: Static	Recover: Assumption	Recover: Integration	Reuse
Autonomous perturbation data	.550	.120/.252	.000/.023	.029/.068	.074/.904
Ours	.700	.594/1.000	.587/.830	.244/.657	.111/.842

A.2.4 Training Details

All models are trained with full-parameter optimization for one epoch using BF16 precision and gradient checkpointing. Each training run uses 30,000 examples. The clean and corrupted SFT datasets contain 30,000 examples each, while the mixed SFT dataset contains 15,000 clean and 15,000 corrupted examples. DPO uses 30,000 preference pairs, and GRPO uses the same 30,000 corrupted-interaction prompts with answer-based rewards. No downstream-reuse examples are included in training.

For SFT, we use a maximum sequence length of 2,048, a learning rate of 1×10^{-5}, the AdamW optimizer, zero weight decay, and a warmup ratio of 0.03. For DPO and GRPO, we use a maximum sequence length of 2,048, with maximum prompt and completion lengths of 1,536 and 512, respectively, and a learning rate of 2×10^{-7}. GRPO samples eight generations per prompt using a temperature of 1.0 and top-p of 1.0.

A.3 Exploratory Auto-Research Experiments

We additionally conducted an exploratory case study inspired by autonomous research. We provided a strong coding agent (GPT-5.6 Sol with Ultra effort) with the same existing generator, initial codebase, and seed tasks used in our study. The agent was given a objective of constructing robustness-training data, but was not instructed which perturbation families to implement. It autonomously inspected the workspace, designed a perturbation pipeline, implemented and validated the generator, and produced 30,000 training examples. We then used the resulting dataset to fine-tune the same 3B base model for one epoch under the otherwise matched SFT setting.

Under the directly comparable Solve setting, training on the autonomously generated data reaches 0.550 accuracy, compared with 0.700 for our structure-aligned mixed data. Its transfer to the other workflows is also limited: end-to-end recovery is 0.120 for static corruption, 0 for unsupported assumptions, and 0.029 for integration failures. In Reuse, it obtains 0.074 accuracy.

A.3.1 Analysis of the Autonomous Perturbation Strategy

The autonomous generator produced the six perturbation strategies summarized in Table 11.

Inspection of the autonomous generator provides a plausible explanation for this transfer pattern. Its perturbations primarily operate at the representational level: they diversify the surface form or presentation of a seed problem while largely preserving its underlying task structure and reasoning process. Although some strategies alter context order, redundancy, or information quantity, none

Table 11: Perturbation strategies designed by the autonomous coding agent.

Strategy	Perturbed aspect	Operation
Query-first shuffle	Information order	Move the question to the beginning and shuffle the order of the supporting evidence or facts.
Symbolic constraints	Expression format	Convert natural-language constraints into symbolic expressions, such as `C[X]=Y`.
Opaque entity aliases	Entity names	Replace the original entity names with opaque identifiers, such as `L1-E1`.
Redundant evidence	Contextual noise	Add repeated facts and require the model to avoid counting duplicated evidence more than once.
Minimal dependency context	Information quantity	Remove facts that are unnecessary for the target dependency graph and retain only the information required for solving the problem.
Compact record notation	Data encoding	Compress natural-language facts into a compact record representation, such as `N[X]=Y`.

introduces an incorrect reasoning state, changes the target dependency structure, or constructs an interaction that requires recovery from and subsequent reuse of a corrupted conclusion. The resulting data therefore primarily constitutes surface/form-level augmentation, rather than task-level or reasoning-level perturbation. It may expose the model to more ways of expressing a problem, but it does not directly supervise the model to recognize an incorrect intermediate state, revise a faulty dependency or assumption, integrate conflicting evidence, or avoid reusing a corrupted conclusion. This distinction offers a consistent explanation for why the autonomously generated data provides some direct-solving utility but transfers poorly to Recover and Reuse.

This case study suggests that satisfying formal data constraints and a broad "improve robustness" objective does not by itself lead an autonomous agent to discover supervision that targets error detection, recovery, and downstream reuse. Explicit decomposition of the intended workflows and reasoning structures remains important when designing robustness-training data. This result should be interpreted as an exploratory study of one agent, prompt, and generation run, rather than as a general comparison of autonomous research systems.

A.3.2 Instruction Given to the Coding Agent

The coding agent received the following instruction, reproduced with its formatting condensed for space. Paths below identify the existing repository and the shared seed-task directory in its workspace.

Autonomous perturbation-generator instruction

```
You are given a workspace containing an existing task-generation
    repository and its source code, together with a directory
    containing the shared initial seed task dataset.

Your goal is to autonomously design and implement a new perturbation-
    task generator based on the materials available in this workspace.
     The purpose of the generated data is to train a model so that it
    becomes more robust to perturbed or challenging task formulations.
     You should independently inspect the existing repository and seed
     dataset, understand how the current task generator works, and
    decide how to modify, extend, or redesign it to produce useful
    robustness-training data.

## Seed-based generation constraint

The provided dataset in iGSM-agent/data_store_train is the source
    dataset for generation, not merely a reference dataset. Every
    final generated training instance must be derived from an instance
     in the provided seed dataset through a transformation or
```

```
    perturbation process. Do not generate independent tasks from
    scratch that are unrelated to the provided seed instances. You are
     free to independently determine how the seed instances should be
    transformed and what perturbation strategies are most useful. The
    design of these perturbation strategies is part of your task.

## Access constraint

You may only use information, files, code, and data available inside
    the iGSM-agent workspace.

## Final dataset requirement

Your final generator must be capable of producing exactly 30,000
    training instances. The final dataset must be a JSON array in
    which every instance contains exactly two fields: instruction and
    output. For every instance, token_count(instruction) + token_count
    (output) <= 2048. The final submitted dataset must contain exactly
     30,000 valid instances.

## Objective

Design the generator with the goal of producing training examples that
     are useful for improving model robustness. The generated tasks
    must remain valid and answerable, and the corresponding output
    must remain the correct target response for the generated
    instruction. The perturbation process should produce meaningful
    modifications of the provided seed tasks rather than simply
    copying the seed instances unchanged. Beyond these requirements,
    you should independently determine perturbation strategies.

## Implementation

Implement the complete generation pipeline in the workspace. You may
    inspect and reuse the existing generator, modify existing code,
    add new modules or scripts, implement filtering or validation, and
     run experiments on the provided seed data. The perturbation
    strategy itself should be designed autonomously. Do not ask the
    user to specify what kinds of perturbations should be used.

## Final deliverables

Leave the workspace in a reproducible state. The deliverables must
    include: (1) the complete source code of the perturbation-task
    generator; (2) a clear executable entry point or command; (3) the
    final dataset containing exactly 30,000 instances; (4) a short
    README describing the approach, seed-data use, execution command,
    required inputs, output location, and important implementation
    decisions; and (5) a validation step checking that the dataset has
     exactly 30,000 instances, every instance contains exactly non-
    empty instruction and output fields, every instance satisfies the
    2048-token combined limit, and the output is valid JSON. The
    result must be reproducible using only files available in the
    workspace.
```