UNDERSTANDING PRE-SCALING REASONING UNDER FALSE PREMISES IN CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Adaptive test-time scaling increasingly relies on instance-level signals to decide where additional reasoning should be spent. Existing allocation strategies commonly use task difficulty, agreement, or verifier feedback, yet misleading specifications introduce a different source of risk: an otherwise solvable instance can fail because a false premise redirects its solution strategy. We study this specification-conditioned vulnerability in code generation and show that it is both recurrent and selective. Across models, benchmarks, and premise families, false premises induce failures in only a minority of baseline-correct instances, leaving substantial heterogeneity within coarse model, dataset, and task groups. Behavioral analysis further shows that vulnerable and robust-correct instances begin with similar rates of Direct Premise Exploitation, then diverge sharply after false-premise injection, with induced failures disproportionately shifting toward direct reliance on the misleading premise. We introduce **Pre-Scaling Reasoning (PSR)**, a design principle for adaptive reasoning allocation that uses pre-generation vulnerability estimates to decide where additional robustness reasoning should be allocated. Under a fixed 20% selection budget, PSR identifies **48.64%** of induced failures, compared with **31.21%** for leave-one-model-out task difficulty. When all routing policies use the same fixed prompting-based mitigation, PSR-guided allocation improves Pass@1 by **+5.77 pp** and exceeds LOMO-guided allocation by **+2.92 pp**. These results establish specification-conditioned vulnerability as a practical pre-generation signal for allocating limited robustness reasoning.

1 INTRODUCTION

Test-time scaling (TTS) improves code generation by investing additional inference computation in candidate generation, verification, and refinement (Li et al., 2022; Brown et al., 2024; Chen et al., 2023; 2024b). As this computation becomes adaptive, the allocation decision becomes increasingly important: how much computation should be spent, and which instances should receive it? Existing approaches use signals such as estimated task difficulty, agreement among sampled answers, and verifier feedback to guide adaptive allocation (Snell et al., 2025; Aggarwal et al., 2023; Wu et al., 2025). These signals primarily characterize how difficult an instance is to solve. A misleading specification introduces another source of risk: an otherwise solvable instance may become vulnerable because the specification steers generation toward an invalid solution strategy. We study this specification-conditioned vulnerability and its implications.

False premises provide a controlled setting for examining this problem. A natural-language specification can assert an input property that does not hold, such as claiming that a list is already sorted (Hu et al., 2023; Yu et al., 2023; Li et al., 2025b). In code generation, such a statement can alter the chosen algorithm while the task objective, function signature, and evaluation tests remain unchanged. We call an instance an *induced failure* when the model solves the original specification correctly and becomes incorrect after the false premise is introduced. As illustrated in Figure 1 (a), task difficulty and false-premise vulnerability can vary independently. The same premise can leave many baseline-correct instances unaffected while redirecting a smaller subset toward failure. This selectivity creates an allocation problem: additional robustness reasoning is most valuable when it can be directed toward the instances that are actually vulnerable. Prior work develops mitigation strategies for false premises and studies how misleading context affects downstream verification (Dhuliawala et al., 2024; Xu & Ma, 2024; Qin et al., 2025; He et al., 2026). In particu-

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

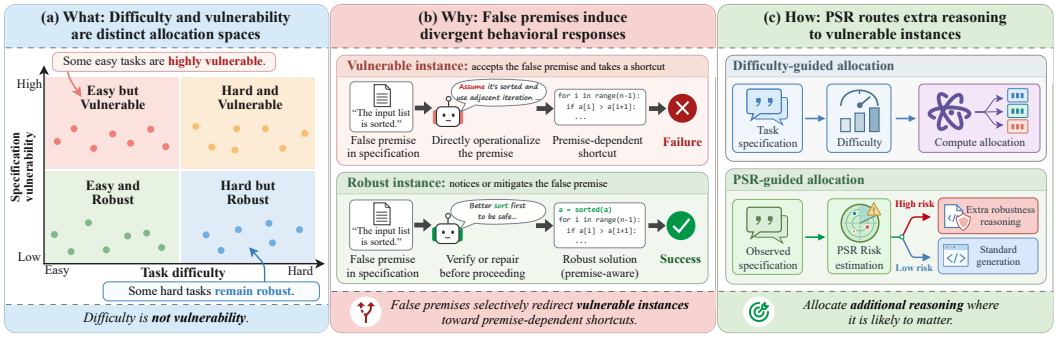

Figure 1: **Overview of Pre-Scaling Reasoning (PSR) under false premises. (a)** Task difficulty and specification vulnerability define distinct allocation dimensions: easy tasks can be highly vulnerable, while hard tasks can remain robust. **(b)** The same false premise can induce divergent behavioral responses: vulnerable instances directly operationalize the misleading premise into brittle shortcuts, whereas robust instances verify or restore the required condition before proceeding. **(c)** PSR allocates additional robustness reasoning according to specification-conditioned vulnerability estimated before generation, while low-risk instances follow standard generation.

lar, recent evidence shows that a false premise can propagate into selection, causing a verifier to prefer a premise-consistent but incorrect solution even when a correct candidate is available (He et al., 2026). We move the allocation decision upstream, before code generation and expensive test-time scaling begin. We introduce **Pre-Scaling Reasoning (PSR)**, a pre-generation decision stage that estimates instance-level vulnerability from specification-visible information and selectively allocates additional robustness reasoning. PSR instantiates this allocation principle upstream of generation and verification: given a limited intervention budget, it prioritizes instances predicted to be vulnerable while leaving lower-risk instances on the standard generation path, as illustrated in Figure 1 (c).

We first ask ***Q1: When is additional robustness reasoning warranted, and what distinguishes vulnerable instances from robust ones?*** Across models and benchmarks, false-premise failures recur consistently while remaining concentrated in a minority of baseline-correct instances. Model, dataset, premise-family, and task-group statistics provide useful population-level signals, yet substantial variation remains within these coarse contexts. We therefore examine how the false premise changes solution behavior. Vulnerable and robust-correct instances exhibit similar rates of *Direct Premise Exploitation* under the original specification, then diverge sharply after false sortedness is introduced. Program-level examples in Figure 1 (b) illustrate the same pattern: vulnerable solutions directly exploit the asserted condition, whereas robust solutions can check or restore the required condition before relying on it. These results connect selective vulnerability to how misleading specification information is incorporated into the generated solution.

We then ask ***Q2: Can vulnerability be predicted before generation, and can this prediction improve reasoning allocation under a fixed budget?*** Using only specification-visible information, our PSR instantiation estimates induced-failure vulnerability before program generation or execution. Under a global 20% selection budget, the highest-risk instances contain **48.64%** of induced failures, compared with 31.21% for leave-one-model-out (LOMO) clean-task difficulty and 23.77% for generic difficulty. The predictive signal remains informative within fixed coarse contexts and partially anticipates Direct Premise Exploitation. We then evaluate whether this signal improves allocation by fixing the same prompting-based mitigation across routing policies and varying only the selected instances. At a 20% intervention-call budget, PSR-guided allocation yields a **+5.77 pp** Pass@1 gain over no intervention, compared with **+2.86 pp** for LOMO-difficulty routing, corresponding to a **+2.92 pp** allocation advantage under the same mitigation and budget. Successful recoveries also align with reduced Direct Premise Exploitation, connecting allocation gains with the identified behavioral vulnerability. These results show that specification-visible information provides a pre-generation vulnerability signal for selective robustness intervention under limited budgets, enabling more targeted allocation of additional reasoning resources across vulnerable instances. Our key contributions are as follows:

❶ We identify a specification-conditioned reasoning-allocation problem under false premises. Controlled false-premise interventions reveal vulnerability across models and benchmarks, while most baseline-correct instances remain robust. Coarse model, premise-family, and task-group priors

provide useful population-level risk signals but leave substantial instance-level heterogeneity. This selectivity motivates allocating additional robustness reasoning to the subset of instances that are most vulnerable to misleading specifications under limited inference budgets.

❷ **We characterize the behavioral structure of selective false-premise vulnerability.** Paired behavioral analyses show that vulnerable and robust-correct instances exhibit similar rates of Direct Premise Exploitation under the original specification, then diverge sharply after false-premise injection. Under false sortedness, Direct Premise Exploitation rises from **6.71% to 90.67%** among induced failures, compared with **5.86% to 35.78%** among robust-correct instances. Cross-model analyses and program-level case studies further show how direct premise exploitation and condition restoration correspond to divergent outcomes under the same misleading premise.

❸ **We formulate Pre-Scaling Reasoning as a design principle for vulnerability-aware inference allocation.** Using only specification-visible information, a lightweight PSR instantiation predicts induced failure before generation and captures **48.64%** of induced failures under a 20% selection budget, compared with **31.21%** for LOMO difficulty. With the same mitigation and budget, PSR-guided routing achieves a **+5.77 pp** Pass@1 gain and exceeds LOMO-guided allocation by **+2.92 pp**. Successful recoveries also align with reduced Direct Premise Exploitation, connecting allocation gains to the identified behavioral vulnerability under misleading specifications.

2 Related Work

False Premises and Unreliable Specifications. False premises arise when a specification asserts a property that does not hold. They have been studied predominantly in question answering, where the central concern is whether a model identifies and refuses such an assumption, and more recently in code generation (Hu et al., 2023; Yu et al., 2023; Li et al., 2025b;c; Kaur et al., 2024). Prior work falls into three broad strands. ♠ ***Failure under false premises***, where studies measure how erroneous assumptions degrade model performance, through benchmarks of presupposition failures, faulty-premise questions, and abstention suites that include them (Yu et al., 2023; Daswani et al., 2024; Shafiei et al., 2025; Kirichenko et al., 2025; Li et al., 2025c; Kaur et al., 2024; Rajpurkar et al., 2018). ♡ ***Recovery from false premises***, where methods surface or correct an unreliable assumption before it reaches the final answer, using prompting-based self-checking, retrieval-augmented premise verification, or decoding-time intervention (Dhuliawala et al., 2024; Qin et al., 2025; Xu & Ma, 2024; Yuan et al., 2024; Lightman et al., 2024). ♢ ***Reasoning under faulty context***, where studies trace how erroneous context reshapes the solution that follows, covering deference to user beliefs, distractibility under irrelevant context, and premise-level chain analyses (Sharma et al., 2024; Shi et al., 2023; Mirzadeh et al., 2025; Mukherjee et al., 2025; Fanous et al., 2025; Sicilia et al., 2025). Across these lines, harm is reported in aggregate: degradation is averaged over tasks, models, and premise families, so a premise that breaks only a few of its tasks is reported like one that breaks most of them.

Test-Time Scaling and Efficient Inference. A growing line of work improves code generation and reasoning by allocating additional computation at test time (Li et al., 2022; Brown et al., 2024; Snell et al., 2025; Wu et al., 2025; Li et al., 2025a). These approaches can be broadly organized into three strategies. ♠ ***Candidate generation and aggregation***, where parallel sampling or self-consistency produces many candidates and aggregates them to raise the chance that at least one is correct (Li et al., 2022; Brown et al., 2024; Wu et al., 2025; Wang et al., 2023; Chen et al., 2024a). ♡ ***Sequential refinement and verification***, where budget is spent after the first attempt, on search over intermediate steps, execution-guided repair, or verifier-based ranking and selection (Yao et al., 2023; Chen et al., 2024b; Olausson et al., 2024; Chen et al., 2023; Ni et al., 2023; Shi et al., 2022; Lightman et al., 2024; Qin et al., 2026; Ma et al., 2026; Shi et al., 2026). ♢ ***Adaptive compute allocation***, where the amount of computation is adjusted per instance using signals such as estimated task difficulty, agreement among sampled answers, or verifier feedback (Snell et al., 2025; Aggarwal et al., 2023; Zhu et al., 2026; Zhang et al., 2025). Across these groups, both the budget and the signal describe the instance as a whole, so a premise that makes an otherwise solvable task fail stays invisible when chosen.

Our Position. False-premise robustness and adaptive test-time scaling leave a common allocation question unresolved: which instances warrant additional robustness reasoning before generation begins? Recent work (He et al., 2026) shows that misleading premises can propagate into downstream verification, causing premise-consistent but incorrect candidates to be favored even when correct candidates are available. We move the decision upstream and estimate specification-conditioned vulnerability before generation. Pre-Scaling Reasoning (PSR) uses this instance-level risk signal to allocate a fixed robustness intervention selectively under a limited budget. This formulation connects

false-premise robustness with adaptive inference by making vulnerability, rather than task difficulty alone, an explicit signal for reasoning allocation.

3 METHOD

3.1 PROBLEM SETUP

We study why the same false premise can strongly harm some model–task instances while leaving others robust. Our focus is instance-level false-premise vulnerability: which model–task–premise instances fail because of the misleading premise, and which remain correct? We define an *induced failure* as an instance that is solved correctly under the original specification but becomes incorrect after the false premise is introduced. This heterogeneity motivates Pre-Scaling Reasoning (PSR), a design principle for allocating additional reasoning selectively. When vulnerable instances form only a small subset, applying additional reasoning to every input is unnecessary and can waste inference budget on already-robust cases. Under PSR, instances are prioritized according to their estimated vulnerability before test-time scaling begins.

3.2 EVALUATION PIPELINE

For each task, we construct a matched pair consisting of the original specification and a false-premise specification. We query the same model on both versions and evaluate the resulting programs with the same execution tests. This paired design allows us to directly compare whether introducing the false premise changes the model's outcome on the same underlying task. An induced failure occurs when the original specification is solved correctly but the matched false-premise specification is not. To ensure a fair comparison, we keep the task objective, function signature, examples, generation instruction, decoding configuration, and evaluation tests fixed across the matched pair. The only controlled change is the localized premise statement in the task specification. All model and dataset comparisons follow the same protocol, so observed differences reflect sensitivity to the premise intervention under otherwise matched conditions.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. We evaluate six code-generation models spanning both closed-source and open-source settings. The closed-source models are Claude Sonnet 4.6, GPT-4o, GPT-4o-mini, and GPT-5.2. The open-source models are Qwen2.5-Coder-14B and Kimi-K2.7-Code. This model set spans multiple model families and capability levels, allowing us to test whether the observed false-premise vulnerability is specific to a particular model family or persists across different deployment settings.

Datasets. We evaluate on three code-generation benchmarks: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2025). HumanEval and MBPP provide standard function-level programming tasks with executable tests, while LiveCodeBench provides a complementary set of more recent programming problems.

Evaluation Metrics. We report five primary metrics: Pass@1, Induced Failure Rate, Direct Premise Exploitation Rate, Recall@20, and Allocation Gain. These metrics characterize overall coding performance, premise-induced vulnerability, premise-dependent behavior, high-risk failure coverage under a fixed selection budget, and the value of risk-guided intervention allocation. We follow the paired evaluation protocol in §3.2. Further experimental details, including metric definitions and statistical procedures, are provided in §S1.

4.2 WHEN IS PRE-SCALING REASONING WARRANTED?

We examine the empirical motivation for selective reasoning allocation in three steps across models, benchmarks, and controlled interventions. **(I) Recurrence.** We test whether false-premise failures recur across models and benchmarks. **(II) Selectivity.** We assess whether these failures affect most baseline-correct instances or remain confined to a minority. **(III) Coarse priors.** We examine what model, premise-family, and task-group summaries reveal about where vulnerability occurs.

❶ **False-premise failures recur across models and benchmarks.** As shown in Table 1, false sortedness reduces Pass@1 in every evaluated combination of model and benchmark. The pattern spans both closed-source and open-source models on HumanEval, MBPP, and LiveCodeBench.

Table 1: **False-sortedness degradation across datasets and models.** Base and False report Pass@1; Induced denotes the fraction of baseline-correct instances that become incorrect after false-sortedness injection. LiveCodeBench uses benchmark-adapted wording.

	HumanEval				MBPP				LiveCodeBench			
Model	**Base**	**False**	**Δ**	**Induced**	**Base**	**False**	**Δ**	**Induced**	**Base**	**False**	**Δ**	**Induced**
Sonnet 4.6	96.34	80.49	-15.85	17.72	63.81	45.14	-18.68	32.93	54.31	54.11	-0.20	8.86
GPT-4o	89.63	75.00	-14.63	21.09	67.32	48.64	-18.68	31.79	45.09	42.69	-2.40	12.89
GPT-4o-mini	87.80	73.78	-14.02	18.75	63.04	49.81	-13.23	27.16	38.88	35.87	-3.01	14.43
GPT-5.2	97.56	74.39	-23.17	24.38	67.70	44.36	-23.35	37.36	71.94	66.93	-5.01	14.21
Qwen2.5-14B	92.07	73.17	-18.90	22.52	63.81	45.14	-18.68	33.54	32.26	30.26	-2.00	14.91
Kimi-K2.7-Code	98.17	87.20	-10.98	11.80	70.43	54.09	-16.34	25.41	61.92	58.72	-3.21	12.30

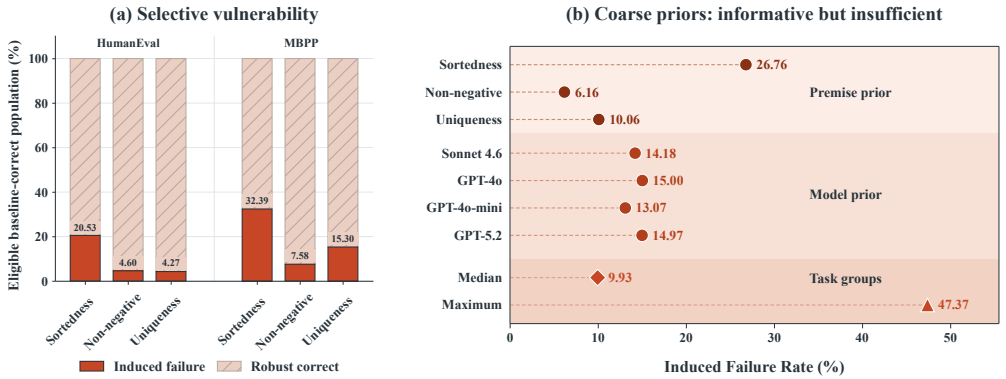

Figure 2: **False-premise vulnerability is selective and cannot be resolved by coarse priors. Left:** Among instances solved correctly under the baseline specification, only a minority become induced failures under a false premise, while most remain robust. **Right:** Premise identity provides useful aggregate risk information, but model-level priors are similar and task-level risk remains highly variable, motivating instance-level risk estimation before generation.

Although the magnitude of degradation varies substantially, its recurrence shows that the problem is not confined to a particular model or benchmark and persists across diverse evaluation settings.

❷ **Vulnerability remains selective across premise families.** Recurrence across settings does not imply that every instance is vulnerable. As shown in Figure 2 (a), most instances solved correctly under the original specification remain correct after the false premise is introduced. This pattern holds across sortedness, uniqueness, and non-negativity, despite substantial differences in induced-failure rates. Across the controlled matrix, every evaluated combination of model, dataset, and premise family retains a robust-correct majority. The need to recover premise-induced failures therefore concerns only a subset of otherwise solvable instances under limited reasoning budgets.

❸ **Coarse priors summarize risk but leave instance selection unresolved.** Figure 2 (b) shows that premise identity provides useful aggregate risk information, while model-level averages are much closer. Task-group rates vary more widely, further indicating that vulnerability is not uniformly distributed. These summaries describe differences in population-level risk, however, rather than identify which individual instances will fail. Vulnerable and robust instances coexist within coarse settings, leaving an instance-level selection problem even when group-level risk is known.

Takeaway 1. Recurring but selective vulnerability motivates Pre-Scaling Reasoning. False-premise failures recur across settings, yet most baseline-correct instances remain robust, and coarse priors summarize risk without resolving individual vulnerability. Under a limited robustness budget, these findings motivate assessing which instances warrant additional reasoning before committing that budget, rather than treating all instances as equally vulnerable.

4.3 What Distinguishes Vulnerable from Robust Tasks?

The same false premise can cause induced failure in some baseline-correct instances while leaving others robust, revealing substantial instance-level heterogeneity under identical specification changes.

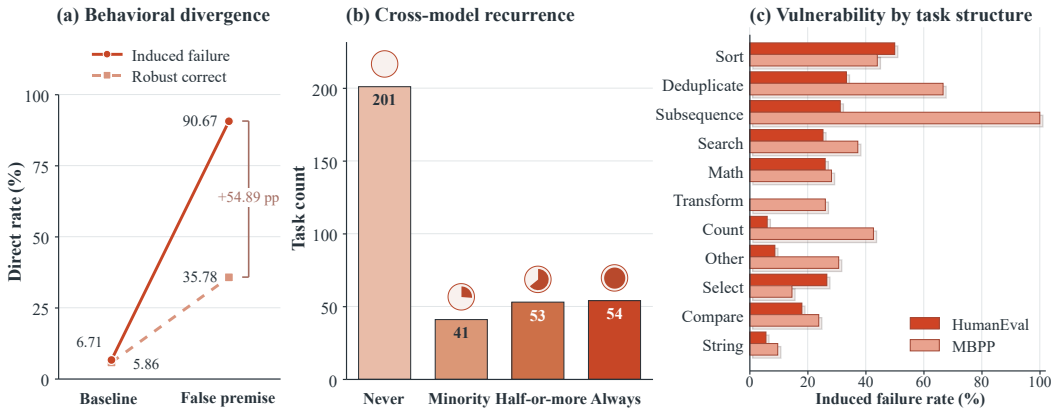

Figure 3: **False-premise vulnerability exhibits behavioral divergence and structural heterogeneity. (a)** Induced failures and robust-correct tasks exhibit similar Direct Premise Exploitation rates at baseline, but diverge sharply after false sortedness is introduced. **(b)** Most tasks never exhibit induced failure across eligible models, while a smaller subset is vulnerable in at least half or all eligible models. **(c)** Induced-failure rates vary substantially across task structures on both HumanEval and MBPP, showing that observed vulnerability is structurally heterogeneous.

We investigate this selective vulnerability at three complementary levels. **(I) Behavioral-level evidence.** We examine how vulnerable and robust-correct instances change their solution behavior after premise injection. **(II) Distribution-level evidence.** We analyze how vulnerability recurs across models and varies across task structures. **(III) Case-level evidence.** We inspect concrete implementations to illustrate how different responses to the same premise lead to different outcomes.

❶ Vulnerable and robust-correct instances diverge in their response to false sortedness. Figure 3 (a) compares Direct Premise Exploitation between induced-failure and robust-correct instances. Their Direct rates are nearly identical under the original specification, but separate sharply after false sortedness is introduced: **Induced Failure:** 6.71% → 90.67% (+83.97 pp); **Robust Correct:** 5.86% → 35.78% (+29.93 pp). The resulting 54.89 pp gap under false sortedness shows that induced failures are accompanied by a substantially stronger shift toward direct premise exploitation.

❷ Vulnerability exhibits cross-model recurrence and variation across task structures. Figure 3 (b) examines whether the same tasks exhibit induced failure across eligible models. Most tasks show no induced failure, while a subset is vulnerable in at least half of the eligible models, including cases where all eligible models exhibit vulnerability. Figure 3 (c) further shows substantial differences in induced-failure rates across task groups on both HumanEval and MBPP. Together, these observations show that vulnerability recurs on particular tasks and that its prevalence varies with task structure, creating substantial instance-level heterogeneity.

❸ Concrete implementations illustrate direct exploitation and condition restoration. Figure 4 traces two behavioral responses to false sortedness. The induced-failure solution applies adjacent iteration directly to the input, relying on the asserted ordering and producing an incorrect result. The robust-correct solution first sorts the input before applying the order-sensitive procedure. By establishing the required condition before relying on it, this solution remains correct despite the misleading premise. The example makes the aggregate behavioral contrast visible in generated code.

Takeaway 2. A stronger shift toward direct premise exploitation characterizes vulnerable instances. Under false sortedness, induced-failure and robust-correct instances diverge sharply from similar baseline Direct rates. Vulnerability also exhibits cross-model recurrence and variation across task structures. The code example illustrates how restoring a required condition can preserve correctness despite the misleading premise.

4.4 Can False-Premise Vulnerability Be Predicted Before Generation?

The behavioral analysis in §4.3 links induced failures under false sortedness to a stronger shift toward Direct Premise Exploitation, revealing a structured behavioral signature of vulnerability. Building on this observation, we ask whether vulnerability can be anticipated *before* code generation using

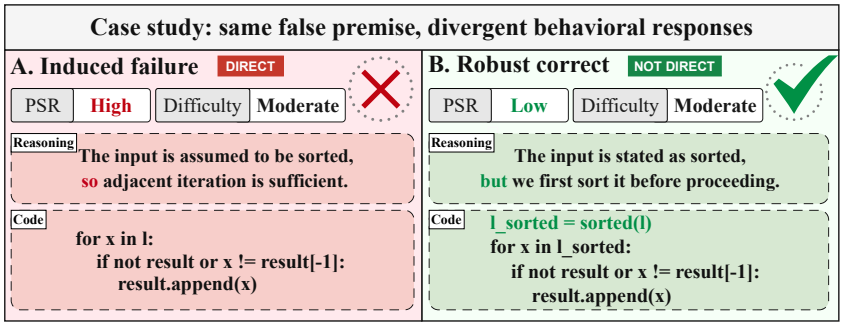

Figure 4: **Case study: the same false premise leads to divergent behavioral responses and outcomes. (a)** The induced-failure solution directly relies on the false sortedness premise through adjacent iteration, producing a brittle premise-dependent implementation. **(b)** The robust-correct solution restores sortedness before using order-sensitive structure, avoiding the same failure.

Table 2: **Pre-generation risk prediction concentrates false-premise vulnerability under a fixed budget.** All predictors use only information available before code generation. Instance-level specification features improve risk discrimination and enable more effective failure capture under a 20% selection budget, supporting selective robustness reasoning allocation with limited resources.

Selection Strategy	AUPRC	AUROC	Failures Captured@20%	Lift@20%
Random Selection	–	–	20.02%	1.000×
Model + Dataset	0.1164	0.5329	21.80%	1.089×
+ Premise Family	0.1857	0.6870	38.87%	1.942×
+ Specification Features	**0.2257**	**0.6992**	**48.64%**	**2.430**×

specification-visible information. We evaluate this question at three complementary levels. **(I) Instance-level evidence.** We test whether specification information improves prediction beyond coarse priors and remains informative within fixed contexts. **(II) Selection-level evidence.** We compare predicted vulnerability with difficulty-based rankings for identifying induced failures under a fixed selection budget. **(III) Behavioral-level evidence.** We examine whether the same pre-generation information also anticipates the premise-exploitation behavior characterized in §4.3.

❶ Specification information predicts vulnerability beyond coarse priors. Table 2 shows progressively stronger prediction as the available information becomes more specific. Model and dataset alone provide limited discrimination. Adding the observed premise family raises AUROC from 0.5329 to 0.6870; incorporating observed-specification features further improves AUPRC (0.1857 → 0.2257) and AUROC (0.6870 → 0.6992). This predictive information also persists within coarse contexts. As shown in Figure 5, the specification-based predictor achieves AUROC above 0.5 in 22 of 24 cells, with 14 exceeding 0.6, when model, dataset, and premise family are held fixed. These results support instance-level predictability within most evaluated contexts using information available before generation, enabling more targeted vulnerability-aware allocation under limited budgets.

❷ Predicted vulnerability captures more failures than difficulty under a fixed budget. We next evaluate how effectively these predictions identify the vulnerable minority when only 20% of eligible instances can be selected. Table 3 compares rankings on the same 3,846 eligible instances, containing 551 induced failures. Under the same global selection budget, PSR captures **48.64%** of induced failures, compared with 31.21% for LOMO difficulty, 23.77% for generic difficulty, and 20.02% for random selection. The evaluated difficulty rankings provide useful risk information, while specification-aware prediction concentrates a larger share of induced failures within the selected subset. Its failure capture reaches **2.43**× that of random selection.

❸ Pre-generation information also anticipates the behavioral signature of vulnerability. We further examine whether the same pre-generation information predicts Direct Premise Exploitation, the behavior associated with induced failure in §4.3. The behavioral diagnostics show that subsequent Direct behavior is partially predictable from specification-visible information, and that induced failures are strongly concentrated among programs exhibiting this behavior (see §S2). Together, these observations suggest that the specification features capture information about susceptibility to premise-

Table 3: **Difficulty-based signals capture only part of false-premise vulnerability.** All routing strategies rank the same 3,846 eligible instances containing 551 induced failures under a fixed 20% intervention budget, highlighting the gap between task difficulty and specification-conditioned risk.

Routing Signal	Failures Captured@20%	Gain over Random	Lift
Random	20.02%	0.00 pp	1.00×
Generic Difficulty	23.77%	+3.75 pp	1.19×
LOMO Difficulty	31.21%	+11.19 pp	1.56×
PSR Vulnerability	**48.64%**	**+28.62 pp**	**2.43×**

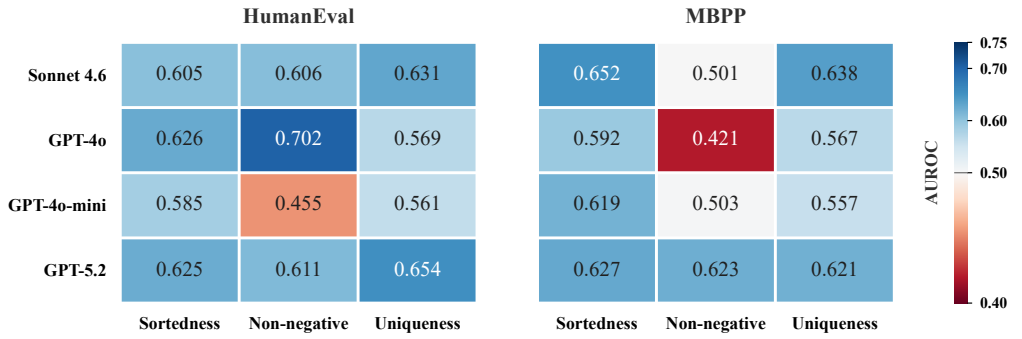

Figure 5: **Within-context predictability persists after fixing coarse factors.** AUROC of the observed-specification predictor is evaluated separately within each fixed model–dataset–premise-family cell. The predictor exceeds chance in 22 of 24 cells and exceeds 0.6 AUROC in 14 cells, showing that instance-level specification information remains informative even after model, dataset, and premise family are held fixed, beyond aggregate population-level risk patterns.

dependent behavioral redirection. This alignment provides a behavioral interpretation of the improved failure localization: the available information partly anticipates the same premise-exploitation pattern that distinguishes vulnerable from robust-correct instances.

Takeaway 3. False-premise vulnerability is partially predictable before generation. Specification information improves prediction beyond coarse priors and remains informative within most fixed contexts. PSR captures more induced failures than the evaluated difficulty rankings, enabling selective robustness intervention under limited inference budgets.

4.5 Can Pre-Generation Risk Guide Reasoning Allocation?

The pre-generation estimates in §4.4 identify instances at elevated risk of induced failure. We test whether this information improves reasoning allocation using a fixed prompting-based mitigation strategy following the protocol in (He et al., 2026). Allocation is evaluated offline on the full routing population of 1,684 model–task instances, with the same mitigation used across policies. We examine this question at three complementary levels. **(I) Recovery-level evidence.** We compare performance recovery across intervention-call budgets. **(II) Allocation-level evidence.** We assess the contribution of instance selection by comparing policies under the same mitigation and a fixed intervention-call budget. **(III) Behavioral-level evidence.** We examine whether successful recovery accompanies a reduction in Direct Premise Exploitation, the behavior associated with vulnerability in §4.3.

❶ **Risk-guided routing improves recovery under limited budgets.** For comparison, leave-one-model-out (LOMO) difficulty estimates clean-task difficulty from the correctness of the other evaluated models on the original specification, excluding the target model's outcomes. As shown in Figure 6 (a), PSR-guided routing achieves greater pass-rate recovery than LOMO difficulty at the same limited intervention-call budgets. Both policies apply the same mitigation to their selected instances, so the curves compare how effectively their rankings prioritize intervention opportunities. At full coverage, the policies coincide because every instance receives the same mitigation.

❷ **Instance selection improves recovery with mitigation and budget held fixed.** Figure 6 (c) compares all four allocation policies at the frozen 20% operating point. PSR achieves a **+5.77 pp** pass-rate gain over no intervention, compared with +3.47 pp for random allocation, +2.86 pp for

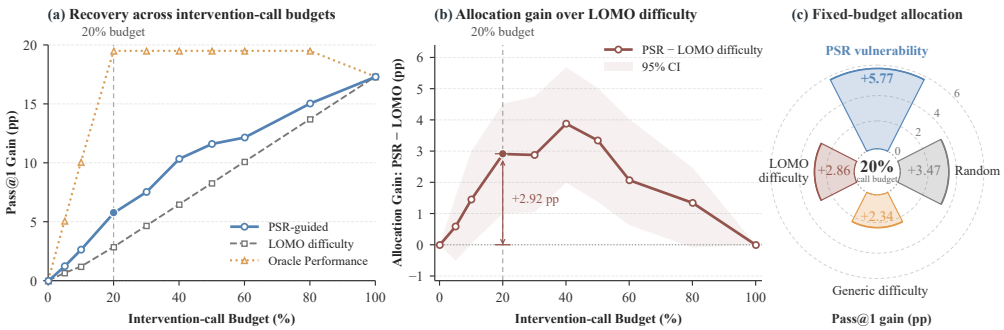

Figure 6: **PSR-guided allocation converts predicted vulnerability into selective recovery. (a)** Under the same preventive intervention, PSR-guided routing recovers Pass@1 faster than LOMO-difficulty routing as the intervention-call budget increases; Oracle Performance provides a post-hoc upper bound. **(b)** PSR provides a **+2.92 pp** allocation gain over LOMO difficulty at the 20% budget, with the advantage vanishing at full coverage. **(c)** At the fixed 20% intervention-call budget, PSR-guided allocation achieves the largest Pass@1 gain among evaluated routing strategies while using the same intervention and routing population.

LOMO difficulty, and +2.34 pp for generic difficulty. The paired advantage over LOMO difficulty is **+2.92 pp**, as shown in Figure 6 (b). With the mitigation and intervention-call budget held fixed, these differences measure the value of selecting instances by predicted vulnerability for this particular intervention, demonstrating the practical benefit of specification-conditioned risk-guided allocation.

❸ **Successful recovery aligns with reduced Direct Premise Exploitation.** We connect these allocation results to the behavioral distinction identified in §4.3. Within the high-risk HumanEval subset, **six of seven recovered instances** transition from Direct to Not Direct behavior after intervention. In contrast, none of the cases that remain Direct recover, while the single instance transitioning to Direct regresses. This pattern aligns successful recovery with reduced reliance on the misleading premise and connects risk-guided intervention to the premise-dependent behavior associated with induced failure, providing additional evidence that vulnerability-aware routing targets the behavioral mechanism underlying selective failures across diverse model settings.

Takeaway 4. Pre-generation risk improves allocation of robustness intervention. PSR-guided routing achieves the largest Pass@1 gain among evaluated policies under the same 20% intervention-call budget. Successful recoveries also align with reduced Direct Premise Exploitation, connecting selective intervention to premise-dependent vulnerability.

5 Conclusion

Conclusion. We study how limited robustness reasoning should be allocated under misleading task specifications. False-premise vulnerability recurs across models and benchmarks, yet remains concentrated in a minority of otherwise solvable instances. Behavioral analysis links this selectivity to stronger Direct Premise Exploitation, while robust solutions can check or restore the required condition. We further show that vulnerability is partially predictable before generation from specification-visible information. Pre-Scaling Reasoning (PSR) provides a design principle for turning this signal into selective allocation, yielding greater recovery than difficulty-based routing under the same fixed mitigation and intervention budget. These results establish specification-conditioned vulnerability as a useful signal for adaptive test-time reasoning.

Limitations. Our study focuses on controlled false premises involving localized input-property assumptions and evaluates a fixed prompting-based mitigation. PSR is introduced as a design principle for vulnerability-aware reasoning allocation rather than an optimized vulnerability predictor. Accordingly, we intentionally use a lightweight specification-based estimator to demonstrate that pre-generation vulnerability can serve as an actionable allocation signal. Developing stronger estimators, broader specification understanding mechanisms, and adaptive interventions remain important directions for future work.

AI USE STATEMENT

Large language models (LLMs) were used to assist with language polishing and portions of code implementation during the preparation of this work. All AI-assisted text and code were reviewed and verified by the authors. The authors take full responsibility for the content of the manuscript, the implementation, and the reported results. The use of LLMs as evaluated models and as part of the experimental methodology is separately documented in the corresponding sections of the paper.

REPRODUCIBILITY STATEMENT

We provide detailed information to support the reproducibility of our experiments in the main paper and supplementary material. The supplementary material specifies the evaluation metrics, the construction and evaluation protocol for the pre-generation vulnerability predictor, the specification-visible features used by the predictor, the fixed-budget routing and intervention procedures, and the statistical analysis. We also document the behavioral classification protocol and provide an independent human validation of the behavioral labels. Together, these details are intended to make the experimental setup and the main analyses reproducible.

ETHICS STATEMENT

This work studies the robustness of code-generation models to misleading task specifications using established code-generation benchmarks and controlled false-premise interventions. The experiments do not involve the collection of personal or sensitive user data or deployment to real users. The human validation reported in §S4.5 is limited to annotating generated code outputs under a fixed behavioral rubric and does not involve personal or sensitive information. We do not identify additional material ethical risks specific to the experimental setting beyond the broader risks associated with the use and deployment of code-generation models.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 12375–12396, 2023.

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

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. In *International Conference on Learning Representations (ICLR)*, 2023.

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

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language models. In *ICML 2024 Workshop on In-Context Learning*, 2024a.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*, 2024b.

Ashwin Daswani, Rohan Sawant, and Najoung Kim. Syn-qa2: Evaluating false assumptions in long-tail questions with synthetic qa datasets. *arXiv preprint arXiv:2403.12145*, 2024.

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

Aaron Fanous, Jacob Goldberg, Ank Agarwal, Joanna Lin, Anson Zhou, Sonnet Xu, Vasiliki Bikia, Roxana Daneshjou, and Sanmi Koyejo. Syceval: Evaluating llm sycophancy. In *AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, pp. 893–900, 2025.

Feng He, Hejia Wang, Linghao Meng, Ming Gao, and Qiankun Li. Verification trap: Understanding test-time selection failures under false premises in code generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Shengding Hu, Yifan Luo, Huadong Wang, Xingyi Cheng, Zhiyuan Liu, and Maosong Sun. Won't get fooled again: Answering questions with false premises. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 5626–5643, 2023.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Navreet Kaur, Monojit Choudhury, and Danish Pruthi. Evaluating large language models for health-related queries with presuppositions. In *Findings of ACL*, pp. 14308–14331, 2024.

Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. Abstentionbench: Reasoning llms fail on unanswerable questions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of EMNLP*, pp. 15964–15978, 2025a.

Jialin Li, Jinzhe Li, Gengxu Li, Yi Chang, and Yuan Wu. Refining critical thinking in llm code generation: A faulty premise-based evaluation framework. *arXiv preprint arXiv:2508.03622*, 2025b.

Jidong Li, Lingyong Fang, Haodong Zhao, Sufeng Duan, and Gongshen Liu. Judge before answer: Can mllm discern the false premise in question? *arXiv preprint arXiv:2510.10965*, 2025c.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Zeyao Ma, Jing Zhang, Xiaokang Zhang, Jiaxi Yang, Zongmeng Zhang, Jiajun Zhang, Yuheng Jing, Lei Zhang, Hao Zheng, Wenting Zhao, et al. Scaling agentic verifier for competitive coding. *arXiv preprint arXiv:2602.04254*, 2026.

Seyed Iman Mirzadeh, Keivan Alizadeh, 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 (ICLR)*, 2025.

Sagnik Mukherjee, Abhinav Chinta, Takyoung Kim, Tarun Anoop Sharma, and Dilek Hakkani-Tür. Premise-augmented reasoning chains improve error identification in math reasoning with llms. In *International Conference on Machine Learning (ICML)*, pp. 45109–45128, 2025.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning (ICML)*, pp. 26106–26128, 2023.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations (ICLR)*, 2024.

Peijia Qin, Ruiyi Zhang, Qi Cao, and Pengtao Xie. Daj: Data-reweighted llm judge for test-time scaling in code generation. *arXiv preprint arXiv:2601.22230*, 2026.

Yuehan Qin, Shawn Li, Yi Nian, Xinyan Velocity Yu, Yue Zhao, and Xuezhe Ma. Don't let it hallucinate: Premise verification via retrieval-augmented logical reasoning. *arXiv preprint arXiv:2504.06438*, 2025.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 784–789, 2018.

Mohammadamin Shafiei, Hamidreza Saffari, and Nafise Sadat Moosavi. Multihoax: A dataset of multi-hop false-premise questions. In *Findings of ACL*, pp. 10169–10187, 2025.

Mrinank Sharma, Meg Tong, Tomek Korbak, David Duvenaud, Amanda Askell, Sam Bowman, Esin Durmus, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, et al. Towards understanding sycophancy in language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3533–3546, 2022.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning (ICML)*, pp. 31210–31227, 2023.

Ji Shi, Peiming Guo, Meishan Zhang, Miao Zhang, Xuebo Liu, Min Zhang, and Weili Guan. Cvedrl: An efficient code verifier via difficulty-aware reinforcement learning. *arXiv preprint arXiv:2601.22803*, 2026.

Anthony Sicilia, Mert Inan, and Malihe Alikhani. Accounting for sycophancy in language model uncertainty estimation. In *Findings of NAACL*, pp. 7866–7881, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

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

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *International Conference on Learning Representations (ICLR)*, 2025.

Nan Xu and Xuezhe Ma. Decoprompt: Decoding prompts reduces hallucinations when large language models meet false premises. *arXiv preprint arXiv:2411.07457*, 2024.

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 (NeurIPS)*, 36:11809–11822, 2023.

Xinyan Yu, Sewon Min, Luke Zettlemoyer, and Hannaneh Hajishirzi. Crepe: Open-domain question answering with false presuppositions. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 10457–10480, 2023.

Hongbang Yuan, Pengfei Cao, Zhuoran Jin, Yubo Chen, Daojian Zeng, Kang Liu, and Jun Zhao. Whispers that shake foundations: Analyzing and mitigating false premise hallucinations in large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2670–2683, 2024.

Kexun Zhang, Shang Zhou, Danqing Wang, William Yang Wang, and Lei Li. Scaling llm inference efficiently with optimized sample compute allocation. In *Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pp. 7959–7973, 2025.

Xiao Zhu, Xinyu Zhou, Boyu Zhu, Hanxu Hu, Mingzhe Du, Haotian Zhang, Huiming Wang, and Zhijiang Guo. Codescaler: Scaling code llm training and test-time inference via reward models. *arXiv preprint arXiv:2602.17684*, 2026.

SUMMARY OF THE APPENDIX

This supplementary contains additional details for ICLR 2027 submission, titled *"Understanding Pre-Scaling Reasoning under False Premises in Code Generation"*. The supplementary is organized as follows:

- §S1 defines the **evaluation metrics** used throughout the paper. It covers outcome and vulnerability, behavioral response, pre-generation risk prediction and ranking, and downstream intervention allocation.
- §S2 details the **pre-generation vulnerability prediction** used to instantiate PSR. It describes the prediction protocol, specification-visible features, fixed-budget risk ranking, within-context evaluation, and information-leakage controls.
- §S3 details the **risk-guided reasoning allocation** experiments. It specifies the pre-generation routing protocol, allocation signals, fixed robustness intervention, budget-sweep analysis, and behavioral diagnostics of recovery.
- §S4 details the **behavioral classification and diagnostics**. It defines Direct Premise Exploitation, specifies the frozen classification protocol, analyzes its association with vulnerability and recovery, and reports independent human validation of the behavioral labels.

S1 Evaluation Metrics

This section defines the evaluation metrics used throughout the paper. The metrics cover four complementary aspects of the analysis: overall task performance and false-premise vulnerability, behavioral response to misleading specifications, pre-generation risk prediction and fixed-budget ranking, and the downstream value of selective intervention. Unless otherwise stated, percentage changes are reported in percentage points (pp).

S1.1 Outcome and Vulnerability Metrics

Let $Y_i^{\text{base}} \in \{0, 1\}$ denote whether instance i is solved correctly under the original specification, and let $Y_i^{\text{false}} \in \{0, 1\}$ denote correctness for the matched false-premise specification.

Pass@1. For an evaluation population $\mathcal{I}$, Pass@1 is the fraction of instances solved correctly:

$$\text{Pass@1} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} Y_i. \tag{1}$$

When comparing the original and false-premise specifications, we report

$$\Delta\text{Pass@1} = \text{Pass@1}_{\text{false}} - \text{Pass@1}_{\text{base}}. \tag{2}$$

A negative value therefore indicates performance degradation after false-premise injection.

Induced Failure. An instance is an induced failure when it is solved correctly under the original specification and becomes incorrect under the matched false-premise specification:

$$I_i = \mathbb{1}\left[Y_i^{\text{base}} = 1 \wedge Y_i^{\text{false}} = 0\right]. \tag{3}$$

Induced Failure Rate. Because premise-induced failure is defined relative to an otherwise successful solution, its denominator contains only baseline-correct instances:

$$\text{IFR} = \frac{\sum_i \mathbb{1}\left[Y_i^{\text{base}} = 1 \wedge Y_i^{\text{false}} = 0\right]}{\sum_i \mathbb{1}\left[Y_i^{\text{base}} = 1\right]}. \tag{4}$$

This metric measures susceptibility to the misleading premise among instances that the model can solve under the original specification.

S1.2 Behavioral Metrics

We use Direct Premise Exploitation as the primary behavioral indicator. Let

$$B_i \in \{0, 1\}, \tag{5}$$

where $B_i = 1$ denotes that the generated solution exhibits Direct Premise Exploitation and $B_i = 0$ denotes Not Direct.

Direct Premise Exploitation Rate. For a behavioral analysis group G, the Direct rate is

$$R_{\text{Direct}}(G) = \frac{1}{|G|} \sum_{i \in G} B_i. \tag{6}$$

Behavioral Shift. For a fixed group G, we measure the change in Direct Premise Exploitation after false-premise injection as

$$\Delta_{\text{Direct}}(G) = R^{\text{false}}_{\text{Direct}}(G) - R^{\text{base}}_{\text{Direct}}(G). \tag{7}$$

This quantity captures how strongly the false premise redirects the generated solution toward direct reliance on the asserted property.

Behavioral Group Gap. To compare induced-failure and robust-correct instances under the same specification condition, we report

$$\text{Gap}_{\text{Direct}} = R_{\text{Direct}}(G_{\text{IF}}) - R_{\text{Direct}}(G_{\text{RC}}), \tag{8}$$

where G_{IF} denotes induced-failure instances and G_{RC} denotes baseline-correct instances that remain correct after false-premise injection.

S1.3 Prediction and Risk-Ranking Metrics

Let s_i denote the pre-generation vulnerability score assigned to instance i, and let I_i denote the induced-failure label defined above. Prediction quality is evaluated using out-of-fold scores.

AUROC. AUROC measures how well the vulnerability score discriminates induced failures from non-induced-failure instances across all possible score thresholds. Equivalently, it measures the probability that a randomly chosen induced failure receives a higher risk score than a randomly chosen negative instance.

AUPRC. AUPRC is the area under the precision–recall curve for induced-failure prediction. We report AUPRC together with AUROC because induced failures constitute a minority of the eligible prediction population, making precision–recall performance informative about concentration of positive cases.

Failures Captured@20. Let $\mathcal{S}_{20}$ denote the subset selected from the highest-ranked 20% of eligible instances under the corresponding ranking signal. Failures Captured@20, equivalently Recall@20, is defined as

$$\text{Capture@20} = \frac{\sum_{i \in \mathcal{S}_{20}} I_i}{\sum_i I_i}. \tag{9}$$

This metric measures what fraction of all induced failures can be localized within the limited selection budget.

Lift@20. We normalize failure capture by the corresponding random-selection baseline:

$$\text{Lift@20} = \frac{\text{Capture@20}_{\text{policy}}}{\text{Capture@20}_{\text{random}}}. \tag{10}$$

A Lift@20 above 1 indicates that the ranking concentrates induced failures more effectively than random selection under the same selection budget.

S1.4 Allocation and Recovery Metrics

For the allocation analysis, let $Y_i^0 \in \{0, 1\}$ denote correctness before the fixed robustness intervention and let $Y_i^{\text{int}} \in \{0, 1\}$ denote correctness after applying the fixed robustness-prompt regeneration. For allocation policy π and intervention budget b, let $w_i^\pi(b) \in [0, 1]$ denote the inclusion weight assigned to instance i. Fractional weights arise when the remaining allocation mass is distributed across instances tied at the selection boundary.

Recovery. A recovery occurs when intervention changes an incorrect outcome into a correct one:

$$\text{Recovery}_\pi(b) = \sum_i w_i^\pi(b)\, \mathbb{1}\left[Y_i^0 = 0 \wedge Y_i^{\text{int}} = 1\right]. \tag{11}$$

Regression. A regression occurs when intervention changes a correct outcome into an incorrect one:

$$\text{Regression}_\pi(b) = \sum_i w_i^\pi(b)\, \mathbb{1}\left[Y_i^0 = 1 \wedge Y_i^{\text{int}} = 0\right]. \tag{12}$$

Net Recovery. The net number of corrected instances is

$$\text{Net}_\pi(b) = \text{Recovery}_\pi(b) - \text{Regression}_\pi(b). \tag{13}$$

Pass@1 Gain. For an allocation cohort of size N, the resulting pass-rate change is

$$\Delta\text{Pass}_\pi(b) = \frac{100}{N} \sum_i w_i^\pi(b) \left(Y_i^{\text{int}} - Y_i^0\right). \tag{14}$$

Equivalently,

$$\Delta\text{Pass}_\pi(b) = 100 \frac{\text{Net}_\pi(b)}{N}. \tag{15}$$

This is the quantity reported as ΔPass in the fixed-budget allocation results and as Pass@1 Gain in the budget-sweep analysis.

Allocation Gain. To isolate the value of one ranking policy relative to another under the same intervention and budget, we define

$$\text{AllocationGain}_{\pi_1 - \pi_2}(b) = \Delta\text{Pass}_{\pi_1}(b) - \Delta\text{Pass}_{\pi_2}(b). \tag{16}$$

For example, the main PSR-versus-LOMO comparison reports

$$\text{AllocationGain}_{\text{PSR}-\text{LOMO}}(20\%), \tag{17}$$

which compares the two routing signals while holding the intervention and intervention-call budget fixed.

S1.5 Metric Summary

Table S1 summarizes the role and denominator of each metric used in the main analyses.

S1.6 Statistical Reporting

All rate differences are reported in percentage points unless otherwise specified. Prediction metrics are computed from out-of-fold risk scores so that each evaluated prediction is produced by a model that was not trained on the corresponding held-out task. For the allocation budget sweep, 95% confidence intervals are obtained from 10,000 task-clustered bootstrap replicates using the frozen routing scores and allocation protocol. Fractional recovery and regression counts can arise from fractional inclusion weights when score ties occur at the routing cutoff. Behavior-specific uncertainty estimates and classification procedures are reported with behavioral analyses in §S4.

Table S1: **Summary of evaluation metrics.** The metrics distinguish performance, false-premise vulnerability, behavioral response, pre-generation risk prediction, and intervention effectiveness.

Metric	Population / Denominator	Interpretation
Pass@1	Evaluated instances	Absolute code-generation correctness.
ΔPass@1	Matched original and false-premise populations	Change in overall performance after false-premise injection.
Induced Failure Rate	Baseline-correct instances	Fraction of originally solved instances that become incorrect after false-premise injection.
Direct Premise Exploitation Rate	Instances in behavioral analysis groups	Frequency of solutions that directly rely on the asserted premise.
AUROC / AUPRC	Eligible prediction instances	Global discrimination quality of the pre-generation vulnerability score.
Failures Captured@20%	All induced failures	Fraction of induced failures contained in the selected top-20% risk subset.
Lift@20%	Relative to random selection	Concentration of induced failures beyond a budget-matched random baseline.
Recovery / Regression	Intervention-selected instances	Correctness improvements and degradations after intervention.
ΔPass (Intervention)	Frozen allocation cohort	Pass-rate improvement of an intervention policy relative to no intervention.
Allocation Gain	Same cohort, intervention, and budget	Difference in intervention gains between two routing policies.

S2 Pre-Generation Vulnerability Prediction

This section describes the lightweight vulnerability estimator used to instantiate the PSR design principle in our experiments. Our PSR instantiation assigns an instance-level risk score using only information available from the observed specification before code generation. Paired generation outcomes are used offline to define the induced-failure target and evaluate prediction quality; they are not inputs to the PSR score or routing decision.

S2.1 Prediction Target and Pre-Generation Protocol

We formulate vulnerability estimation as binary prediction over model–task–premise instances. An *induced failure* is defined by a paired correct-to-incorrect transition: the model solves the original specification correctly but fails after the false premise is introduced. The frozen prediction evaluation contains 3,846 instances, including 551 induced failures.

At inference time, PSR does not require the paired outcome used to construct this offline label. Instead, the observed specification is directly mapped to a vulnerability score before code generation. Instances can therefore be ranked for selective robustness intervention without first generating or executing a candidate solution.

We use a class-balanced logistic-regression estimator with L1 regularization. The predictor takes the 23 specification-visible features described in Table S3. The three count features are standardized using statistics fitted on the training partition, while the remaining binary features are passed through unchanged.

Prediction is evaluated using five-fold out-of-fold estimation grouped by task ID. For each fold, feature preprocessing and model fitting are performed using only the corresponding training partitions, and the fitted pipeline is applied to the held-out tasks. The resulting held-out induced-failure probabilities are concatenated to form the PSR risk scores used in the prediction analyses.

S2.2 Specification-Visible Features

The PSR predictor uses a lightweight representation of the observed specification. Its 23 features are organized into four groups: text statistics, operation cues, input-modality cues, and task/premise cues. All features are deterministically extracted from text available before generation.

Table S2: **Prediction and pre-generation scoring protocol.** PSR computes vulnerability from specification-visible information before code generation; paired outcomes are used offline to define and evaluate the prediction target.

Component	Protocol
Prediction target	Induced failure derived offline from paired original- and false-premise outcomes
Evaluation instances	3,846 model–task–premise instances
Positive instances	551 induced failures
Predictor input	23 specification-visible features available before generation
Estimator	Class-balanced logistic regression with L1 regularization
Preprocessing	Count features standardized on the training partition; binary features unchanged
Evaluation protocol	Five-fold out-of-fold prediction grouped by task ID
Risk score	Held-out predicted probability of induced failure
Scoring time	Before code generation
Post-generation signals	Not used by the PSR score or routing decision

Table S3: **Specification-visible feature groups used by PSR.** All features are deterministically extracted from the observed specification before code generation.

Feature Group	#	Description
Text Statistics	3	Specification length and example-related statistics.
Operation Cues	12	Lexical indicators of operations such as search, sorting, comparison, selection, and transformation.
Input-Modality Cues	5	Lexical indicators of input structures including lists, strings, numbers, matrices, and graphs.
Task / Premise Cues	3	Lexical indicators related to uniqueness, sortedness, and sign constraints.
Total	**23**	

The three text-statistic features measure basic properties such as specification length and the presence of example-like text. The remaining twenty features are binary lexical indicators obtained using fixed case-insensitive pattern matching. They capture surface cues associated with the operation requested by the task, the apparent input modality, and premise-relevant properties such as sortedness, uniqueness, or sign constraints. These indicators are used as specification cues rather than semantic verification of whether a property actually holds.

S2.3 Fixed-Budget Risk Ranking

We evaluate whether the resulting risk score is useful for prioritizing instances under a limited selection budget. For the prediction analysis in the main text, instances are ranked by their corresponding signal and evaluated at a fixed 20% operating point. We compare PSR against random selection and two clean-task difficulty signals. Generic difficulty estimates the probability of failure on the original task using an out-of-fold predictor. Leave-one-model-out (LOMO) difficulty estimates task difficulty using the original-specification correctness of the other evaluated models while excluding the target model. PSR instead ranks instances by predicted susceptibility to the false-premise intervention. The four signals are summarized in Table S4. For ranking quality, we report AUPRC and AUROC together with two budget-specific measures. *Failures Captured@20* is the fraction of induced failures contained in the highest-risk 20% of instances, and *Lift@20* normalizes this value by the corresponding random-selection coverage.

S2.4 Within-Context Prediction

The pooled prediction results may partly reflect differences across models, datasets, or premise families. We therefore additionally evaluate the specification-based predictor within each fixed model–dataset–premise-family context. Within each cell, AUROC is determined only by instance-level variation after the corresponding coarse factors are held fixed.

Table S4: **Signals used in the fixed-budget ranking comparison.** Difficulty-based signals characterize clean-task performance, while PSR estimates susceptibility to the false-premise intervention.

Signal	Estimated Quantity	Definition
Random	None	Budget-matched random selection.
Generic Difficulty	Clean-task failure risk	Out-of-fold estimate of failure under the original specification.
LOMO Difficulty	Clean-task difficulty	Original-specification correctness aggregated across other evaluated models, excluding the target model.
PSR Vulnerability	Induced-failure risk	Pre-generation vulnerability estimated from specification-visible information.

Table S5: **Within-context AUROC of the specification-based vulnerability predictor.** Prediction is evaluated separately within each fixed model–dataset–premise-family context.

	HumanEval			MBPP		
Model	**Sortedness**	**Non-negativity**	**Uniqueness**	**Sortedness**	**Non-negativity**	**Uniqueness**
Sonnet 4.6	0.605	0.606	0.631	0.652	0.501	0.638
GPT-4o	0.626	0.702	0.569	0.592	0.421	0.567
GPT-4o-mini	0.585	0.455	0.561	0.619	0.503	0.557
GPT-5.2	0.625	0.611	0.654	0.627	0.623	0.621

The complete results are reported in Table S5. The predictor achieves AUROC above 0.5 in 22 of 24 evaluated contexts, with 14 contexts exceeding 0.6. This pattern supports the main-text observation that specification-visible information remains predictive within most coarse contexts.

S2.5 Information Availability and Leakage Control

PSR is evaluated as a pre-generation allocation principle. Therefore, any information used by the PSR instantiation must be available before candidate generation and execution. Table S6 summarizes the information availability during PSR instantiation. The vulnerability estimator receives only specification-visible information and controlled metadata available before generation. It does not access generated programs, execution outcomes, behavioral annotations, or intervention results. Premise category information is available in our controlled evaluation setting and serves as optional metadata for vulnerability estimation. The specification-only instantiation of PSR does not rely on post-generation signals. The induced-failure labels are only used offline to define the prediction target and evaluate the vulnerability estimator. They are not available during deployment-time scoring or routing. In practical deployment scenarios, premise-category information can be obtained from lightweight specification parsing or omitted when unavailable. In our controlled evaluation, we include this metadata to isolate the value of specification-conditioned vulnerability estimation.

S3 Risk-Guided Reasoning Allocation

This section provides additional details on the reasoning-allocation experiments in §4.5. All instances in the allocation population are scored before code generation, and the available intervention budget is assigned according to the resulting ranking. Across allocation policies, the intervention procedure and intervention-call budget are held fixed; only the signal used to prioritize instances changes.

S3.1 Pre-Generation Routing and Fixed Intervention

The allocation study uses the complete matched false-sortedness routing population of 1,684 model–task instances. Each instance receives a routing score using information available before code generation. At the primary operating point, the highest-ranked 20% of the population is assigned to the robustness intervention, corresponding to 337 intervention calls. For tied scores at the selection boundary, the remaining allocation mass is distributed uniformly within the tied block. Random allocation is defined analogously using equal inclusion probability under the same total intervention mass. Selected instances use a frozen robustness-prompt generation with the same target model, while unselected instances follow the standard generation path. The robustness prompt encourages the model to avoid relying on unverified structural assumptions while preserving the original task

Table S6: **Information availability for the PSR instantiation.** PSR uses only information available before generation.

Information Source	Used by PSR
Observed specification	✓
Premise family information	✓
Generated program	–
Execution result	–
Hidden test outcome	–
Induced-failure label	–
Behavioral annotation	–
Intervention outcome	–

Table S7: **Pre-generation routing and intervention protocol.** All allocation policies operate on the same routing population and use the same fixed robustness intervention.

Component	Protocol
Allocation population	1,684 model–task instances
Premise family	False sortedness
Routing scope	Global pooled allocation
Routing time	Before code generation
Main operating point	20% intervention-call budget
Intervention calls at 20%	337
Intervention path	Fixed robustness-prompt generation
Standard path	Standard generation
Target model	Same model for standard and intervention paths
Budget unit	Intervention calls
Token/FLOP matching	No explicit token/FLOP normalization

objective and required function interface. The same intervention is used for every allocation policy. The PSR routing score uses no generated program, execution outcome, verifier score, or behavioral annotation from the routed instance. The budget is measured in intervention calls. We therefore compare policies under the same number of intervention opportunities without claiming exact token- or FLOP-level cost matching. The complete protocol is summarized in Table S7.

S3.2 Allocation Signals

We compare four signals under the same routing population, intervention procedure, and intervention-call budget. Random allocation provides a signal-free reference. Generic difficulty estimates clean-task failure risk using an out-of-fold predictor. Leave-one-model-out (LOMO) difficulty estimates task difficulty from the original-specification performance of the other evaluated models. PSR vulnerability ranks instances by predicted susceptibility to the false-premise intervention using the pre-generation risk score described in §S2. The four signals are summarized in Table S8.

For a target model m and task t, LOMO difficulty is defined as

$$d_{\mathrm{LOMO}}(t, m) = \frac{1}{|\mathcal{M} \setminus \{m\}|} \sum_{m' \neq m} \left(1 - Y_{\mathrm{clean}}(t, m')\right), \tag{18}$$

where $Y_{\mathrm{clean}}(t, m') \in \{0, 1\}$ denotes correctness under the original specification for another evaluated model. The target model's own outcome is excluded from its LOMO score.

Generic difficulty and LOMO therefore characterize clean-task difficulty, whereas PSR estimates susceptibility to a misleading specification. The comparison tests whether specification-conditioned vulnerability provides a useful allocation signal beyond clean-task difficulty.

S3.3 Budget Sweep and Statistical Analysis

We additionally evaluate allocation across the intervention-call budget grid used in Figure 6: 0%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 80%, and 100%. At each operating point, PSR and LOMO rank the same routing population and use the same fixed robustness intervention.

Table S8: **Signals used for intervention allocation.** The policies share the same intervention and budget and differ only in the signal used to prioritize instances.

Signal	Estimated Quantity	Definition
Random	None	Budget-matched random allocation.
Generic Difficulty	Clean-task failure risk	Out-of-fold estimate of failure under the original specification.
LOMO Difficulty	Clean-task difficulty	Original-specification performance aggregated across other models, excluding the target model.
PSR Vulnerability	Induced-failure risk	Pre-generation vulnerability estimated from specification-visible information.

Table S9: **Operational behavioral categories used in the main analyses.** Direct denotes explicit use of the asserted sorted-order property without establishing the required ordering condition.

Behavior	Operational Definition
Direct	The generated solution exhibits explicit use of sorted-order structure without first establishing the required ordering condition.
Not Direct	Cases that do not exhibit Direct behavior under the frozen behavioral rubric.

Figure 6 (a) reports the Pass@1 gain relative to standard generation. Oracle Performance provides a post-hoc diagnostic upper bound based on the realized benefit of the intervention. It is used only to characterize the recoverable headroom of the fixed intervention and is not available to the routing policies.

Figure 6 (b) reports the allocation difference

$$\Delta_{\text{alloc}}(b) = \Delta\text{Pass}_{\text{PSR}}(b) - \Delta\text{Pass}_{\text{LOMO}}(b), \tag{19}$$

where b denotes the intervention-call budget.

Confidence intervals are obtained from 10,000 task-clustered bootstrap replicates. The curves use the stored operating points directly; no intermediate values are obtained by interpolation. At full coverage, the ranking policies coincide because every instance is assigned to the same fixed intervention.

S3.4 Behavioral Alignment of Recovery

We use a separate frozen behavioral subset to examine how successful intervention relates to Direct Premise Exploitation. The diagnostic contains 33 high-risk GPT-4o-mini HumanEval sortedness instances. For each instance, the standard and robustness-prompt generations are categorized as Direct or Not Direct, where Direct denotes explicit reliance on the misleading sortedness premise.

This behavioral analysis is evaluated after generation and is not used by the routing policy. It provides a retrospective interpretation of the recovered cases. As reported in the main-text behavioral transition analysis, recoveries concentrate among instances that move from Direct to Not Direct behavior, linking successful intervention to reduced reliance on the misleading premise. We interpret this pattern as behavioral alignment rather than a causal mediation result.

S4 Behavioral Classification and Diagnostics

S4.1 Direct Premise Exploitation

We use **Direct Premise Exploitation** as an operational behavioral indicator for the false-sortedness setting. A generated solution is labeled *Direct* when it relies on sorted-order structure, such as an order-sensitive adjacent comparison, two-pointer procedure, or binary-search-style strategy, without first establishing the required ordering condition. For the binary analyses reported in the paper, all remaining valid behavioral classifications are grouped as *Not Direct*. This complement is used as a measurement category and does not by itself imply correctness or robustness.

All behavioral analyses use this frozen binary definition consistently: solutions satisfying the Direct criterion are labeled *Direct*, and all remaining valid classifications are grouped as *Not Direct*. The operational categories are summarized in Table S9.

Table S10: **Behavioral classification protocol.** Outcome-derived information is excluded from the classifier input.

Component	Frozen Setting
Classifier	GPT-4o-mini
Temperature	0
Max output tokens	128
Generated code	Included
Generated explanation	Behavioral contrast description only
Execution outcome	Excluded
Hidden-test information	Excluded
Induced-failure label	Excluded
Pre-generation risk score	Excluded
Recovery outcome	Excluded
Behavioral output	Direct / Not Direct

Table S11: **Pre-generation prediction of subsequent Direct Premise Exploitation.** The predictor uses only specification-visible information available before code generation.

Target	AUPRC	AUROC	Recall@20%	Precision@20%	Lift@20%
Direct Premise Exploitation	0.7264	0.7106	30.73%	81.53%	1.536×

S4.2 Behavioral Classification Protocol

We classify generated solutions using a frozen GPT-4o-mini behavioral judge with a fixed rubric. Each candidate is classified independently. The classifier does not receive execution correctness, hidden-test outcomes, induced-failure labels, risk scores, or recovery outcomes. The frozen configuration is summarized in Table S10. The main vulnerable-versus-robust analysis uses the generated program together with its generated explanation, while the intervention-transition analysis applies the same behavioral rubric to generated code. The classifier operates at temperature 0 with a maximum output length of 128 tokens. The behavioral judge therefore characterizes properties of the generated solution independently of the downstream correctness grouping used in our analysis.

S4.3 Behavioral Signature of Vulnerability

The frozen population contains 1,282 model–task pairs across four models, including 343 induced failures and 939 robust-correct instances. Original and false-premise generations are classified independently and are paired only after behavioral labels have been obtained. As reported in the main text, the two groups have similar Direct rates under the original specification, but diverge strongly after false-sortedness injection. Induced failures increase from 6.71% to 90.67% Direct, whereas robust-correct instances increase from 5.86% to 35.78%. This analysis therefore measures a differential behavioral response to the same controlled premise intervention. We further test whether this behavioral response can itself be anticipated before generation. Using the same 23 specification-visible features used for pre-generation risk analysis, we train a lightweight L1-regularized logistic regression predictor to estimate whether the subsequently generated false-premise solution will be classified as Direct. Predictions are evaluated out of fold with task-grouped five-fold cross-validation. The evaluation contains 1,684 model–task instances, of which 894 are Direct. As summarized in Table S11, the predictor reaches AUPRC 0.7264 and AUROC 0.7106, with bootstrap intervals of $[0.6767, 0.7761]$ for AUPRC and $[0.6717, 0.7470]$ for AUROC. These results show that specification-visible information contains predictive signal about the subsequent Direct behavior observed after generation.

S4.4 Behavioral Alignment with Recovery

We finally examine whether changes in the Direct indicator align with intervention outcomes. This analysis uses the frozen-risk top 20% subset of the GPT-4o-mini HumanEval false-sortedness intervention cohort, yielding 33 high-risk instances from a source population of 164 tasks. The pre-intervention behavior is measured on the original false-premise generation, and the post-intervention behavior is measured on the corresponding robustness-regenerated output.

Table S12: **Behavior-transition definitions used in the recovery analysis.**

Transition	Interpretation
Direct → Not Direct	The generated behavior changes from exhibiting Direct behavior to not exhibiting Direct behavior after intervention.
Direct → Direct	The generated behavior continues to exhibit Direct behavior after intervention.
Not Direct → Not Direct	The generated behavior remains outside the Direct category after intervention.
Not Direct → Direct	The generated behavior changes to exhibit Direct behavior after intervention.

We define four behavioral transitions according to the Direct indicator (Table S12).

Within this high-risk subset, six of seven recovered instances undergo a Direct→Not Direct transition. The transition occurs in 85.71% of recovered instances compared with 11.54% of non-recovered instances, corresponding to a difference of **+74.18 pp** with a 95% confidence interval of **[+39.51, +96.43]**. These results provide behavioral alignment between reduced Direct Premise Exploitation and successful recovery. We interpret this relationship as an association rather than causal mediation.

S4.5 Human Validation of Behavioral Labels

To assess the reliability of the behavioral labels used in the main analysis, we conducted an independent human annotation audit of the Direct Premise Exploitation categorization. Two annotators independently reviewed 168 instances sampled from the behavioral evaluation pool under the frozen binary behavioral rubric and assigned Direct or Not Direct labels. The two annotators agreed on 149 of the 168 instances, corresponding to an agreement rate of 88.69%. The resulting Cohen's κ is 0.6965 with a 95% confidence interval of [0.5583, 0.8151]. These results indicate substantial agreement between independent human judgments under the frozen behavioral rubric and support the consistency of the behavioral categorization used throughout the analysis.