TurnBench-MS: A Benchmark for Evaluating Multi-Turn, Multi-Step Reasoning in Large Language Models

Anonymous ACL submission

Abstract

Despite impressive advances in large language models (LLMs), existing benchmarks often focus on single-turn or single-step tasks, failing to capture the kind of iterative reasoning required in real-world settings. To address this limitation, we introduce **TurnBench**, a novel benchmark that evaluates multi-turn, multi-step reasoning through an interactive code-breaking task inspired by a "Turing Machine Board Game." In each episode, a model must uncover hidden logical or arithmetic rules by making sequential guesses, receiving structured feedback, and integrating clues across multiple rounds. This dynamic setup requires models to reason over time, adapt based on past information, and maintain consistency across steps—capabilities underexplored in current benchmarks. TurnBench includes two modes: *Classic*, which tests standard reasoning, and *Nightmare*, which introduces increased complexity and requires robust inferential chains. To support fine-grained analysis, we provide ground-truth annotations for intermediate reasoning steps. Our evaluation of state-of-the-art LLMs reveals significant gaps: GPT-4-mini achieves 81.5% accuracy in Classic mode, but performance drops to 17.8% in Nightmare mode. In contrast, human participants achieve 100% in both, underscoring the challenge TurnBench poses to current models. By incorporating feedback loops and hiding task rules, TurnBench reduces contamination risks and provides a rigorous testbed for diagnosing and advancing multi-step, multi-turn reasoning in LLMs.[1]

1 Introduction

Reasoning is central to human cognition and a key benchmark for evaluating the capabilities of artificial intelligence (AI) systems (Wason and Johnson-Laird, 1972; Dunbar and Klahr, 2012). In the context of large language models (LLMs), assessing reasoning ability is especially critical as these models are increasingly deployed in complex, real-world tasks. While a growing body of work has proposed datasets and evaluation methods for probing LLM reasoning (Zeng et al., 2024; Wang et al., 2023a; Welleck et al., 2022), significant gaps remain in how we measure and interpret this ability—particularly in multi-step, multi-turn settings.

First, most existing evaluations focus on single-turn or single-step reasoning tasks, overlooking the iterative and interactive nature of real-world problem-solving. Human reasoning often involves cycles of information gathering, hypothesis testing, and adaptation to feedback. This is especially true in scenarios where information is incomplete or distributed across multiple interactions. While recent benchmarks attempt to assess multi-step reasoning (Tang et al., 2025; Zeng et al., 2024), they rarely simulate settings that require reasoning across multiple turns.

Second, current evaluation metrics typically emphasize final-answer correctness, with little insight into the model's intermediate reasoning process (Zhuang et al., 2023; Hao et al., 2024). As complex reasoning often admits multiple valid paths, simply scoring final outputs fails to distinguish between genuine inference and lucky guesses. Though some methods attempt process-level evaluation via manual annotation or automated proxies (Zeng et al., 2024; Tang et al., 2025), these are limited by subjectivity and the absence of reliable ground truth for intermediate reasoning.

Third, data contamination poses a serious concern. Static benchmarks—often sourced from public datasets or templated questions—can overlap with pretraining corpora, making it difficult to disentangle memorization from actual reasoning (Yang et al., 2025; Jain et al., 2024; Li et al., 2023). This undermines the reliability of benchmark results and inflates perceived model performance.

To address these gaps, we introduce **TurnBench**, a novel benchmark designed to evaluate multi-turn,

[1] See our code at: https://anonymous.4open.science/r/TurnBench-2552/

multi-step reasoning through an interactive code-breaking task inspired by the *Turing Machine* board game. In this game, a model must uncover a hidden three-digit code by engaging in multiple rounds of interaction with logical verifiers. Each verifier is governed by a hidden rule; only one rule per verifier is active in a given instance. To succeed, the model must iteratively guess codes, select verifiers, analyze feedback, and gradually infer the underlying logical or arithmetic constraints—mirroring how humans perform exploratory reasoning.

TurnBench explicitly addresses key shortcomings in existing benchmarks. First, it evaluates **multi-turn, multi-step reasoning** by requiring LLMs to adapt dynamically to feedback across multiple rounds and integrate partial clues to formulate and revise hypotheses over time. Second, it enables **process-level evaluation** through a rule-based mechanism that compares models' intermediate inferences—i.e., their identification of active rules in each verifier—against ground truth, allowing structured analysis of reasoning steps beyond final answer correctness. Finally, TurnBench offers strong **contamination resistance** due to its dynamic rule configurations: even under fixed game setups, varying rule activations lead to distinct reasoning trajectories, minimizing the risk of data leakage from LLM pretraining corpora. Our work makes the following key contributions:

- We propose **TurnBench**, the first benchmark designed to evaluate *multi-turn, multi-step* reasoning in LLMs through dynamic, interactive tasks. TurnBench includes 540 game instances across two modes—*Classic* and *Nightmare*—with three difficulty levels each.
- We introduce a novel, automated evaluation method that leverages rule-based feedback to analyze intermediate reasoning steps, offering a grounded way to assess the internal thinking of LLMs.
- We benchmark a range of open-source and proprietary models, including GPT-o4-mini and Gemini 2.5 Flash, alongside human participants. Results show a significant performance gap between humans (100%) and models (as low as 17.8% in Nightmare mode), highlighting the challenge TurnBench presents.
- We release a new dataset comprising not only game settings and final answers, but also detailed interaction logs and annotated reasoning steps for both models and humans, providing a valuable resource for future research.

2 Related Work

LLMs in Interactive Game Environments: Recent work has explored the use of LLMs as agents in interactive games to assess their planning, reasoning, and decision-making capabilities across diverse domains such as board games, card games, and social deduction settings (Schultz et al., 2024; Xu et al., 2023; Akata et al., 2023; Light et al., 2023; , FAIR; Wang et al., 2023b; Zhuang et al., 2025; Tang et al., 2025). These benchmarks typically present the game state in textual or structured formats and prompt LLMs to make the next move using natural language generation. For instance, PokerBench (Zhuang et al., 2025) adopts classification-based decision scenarios, while AvalonBench (Light et al., 2023) and BALROG (Paglieri et al., 2025) evaluate agents through multi-turn, interactive gameplay. Common evaluation metrics include win rate, legality of actions, strategy optimality, and task completion.

Benchmarks for Multi-Step and Logical Reasoning: To more directly evaluate reasoning capabilities, recent studies have proposed benchmarks focused on multi-step logical and mathematical inference. Multi-LogiEval (Patel et al., 2024) and Belief-R1 (Wilie et al., 2024) reveal that even advanced LLMs like GPT-4 and Claude struggle with tasks involving deep reasoning and belief revision. MuSR (Sprague et al., 2023) embeds multi-step logic challenges in long-form narratives to test long-context reasoning. Other efforts leverage real-world tasks such as mathematics competitions (e.g., AIME (AoPS, 2024)) to benchmark high-level mathematical reasoning. CriticBench (Lin et al., 2024) and MR-Ben (Zeng et al., 2024) further highlight the potential of multi-round, self-reflective prompting for improving LLM reasoning through iterative critique and correction.

Rule-Based Inference and Tool-Augmented Reasoning: Several benchmarks focus on rule-based or structured inference tasks. GridPuzzle and PuzzleEval (Tyagi et al., 2024) utilize logic grid puzzles, while ZebraLogic (Lin et al., 2025) frames reasoning as constraint satisfaction problems (CSPs). RuleArena (Zhou et al., 2024) evaluates models on dynamic policy reasoning. Tool-

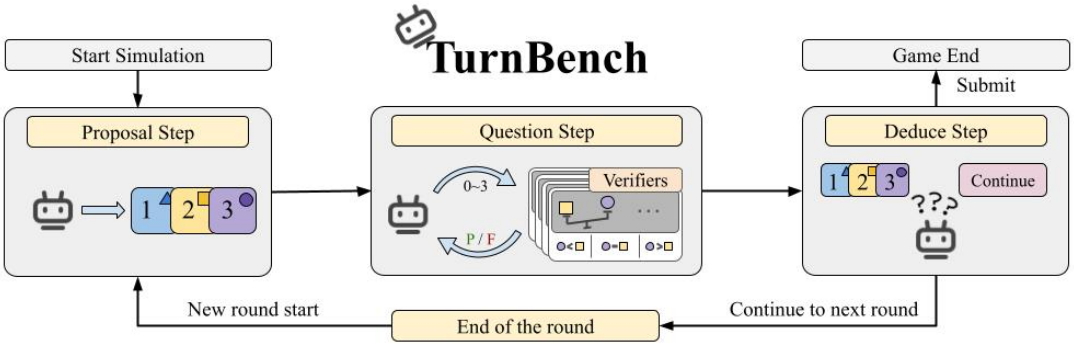

Figure 1: Overview of the TurnBench game framework. The game consists of three iterative stages: Proposal Step, where the LLMs proposes a candidate 3-digit code (represented by colored slots); Question Step, where the agent selects up to three verifiers to evaluate the code and receive Pass/Fail feedback; and Deduce Step, where the agent reasons over verifier responses to decide whether to Submit the code or Continue to the next round with a revised proposal. The loop continues until the agent successfully deduces and submits the correct code..

augmented frameworks like LINC (Olausson et al., 2023) and MATHSENSEI (Das et al., 2024) enable LLMs to perform formal reasoning through external tools. Meanwhile, self-reflection strategies such as Self-Refine (Madaan et al., 2023) and ReFlexion (Shinn et al., 2023) allow models to iteratively revise incorrect or incomplete outputs via internal critique loops.

While the above efforts have made significant strides in evaluating LLM reasoning, several important gaps remain. First, few benchmarks explicitly evaluate *multi-step reasoning across multiple interaction rounds*—a critical feature of real-world problem-solving. Most logic and tool-based tasks are static, single-shot evaluations that do not require models to gather and integrate information over time. Second, existing benchmarks often lack *ground truth for intermediate reasoning steps*, limiting analysis to final-answer accuracy. This makes it difficult to determine whether a correct answer results from genuine reasoning or chance. Third, many datasets are vulnerable to *data contamination* due to overlap with pretraining corpora. Finally, while game-based settings are promising, they rarely focus on rule-discovery and hypothesis refinement under feedback constraints.

TurnBench is designed to fill these gaps by offering a dynamic, interactive benchmark that simulates real-world multi-turn reasoning. It provides ground-truth annotations for intermediate reasoning, minimizes contamination risk through dynamic rule configurations, and emphasizes logical consistency and rule inference across turns.

3 TurnBench

3.1 Turing Machine Game Mechanics

Turing Machine is a logic-based deduction game where the player's objective is to identify a unique three-digit code (digits 1–5), each digit associated with a distinct color (e.g., blue, yellow, purple). The challenge lies in interacting with a set of 4–6 verifiers, each governed by a single, hidden active criterion selected from a predefined rule pool. Players must deduce these hidden criteria and submit a code that satisfies all of them.

Each game unfolds in multiple rounds with four key phases: First, the player composes a proposed code (e.g., BLUE=2, YELLOW=4, PURPLE=3), which remains fixed for the current round. Next, the player queries up to three verifiers sequentially, each returning a binary judgment (PASS/FAIL) based on the verifier's active rule. Using this feedback, the player can either attempt a final answer or skip and continue to the next round for further testing. The game ends once a final answer is submitted.

TurnBench supports two game modes: **Classic** and **Nightmare**, each with Easy, Medium, and Hard difficulty levels. In Classic mode, verifier responses correspond directly to the selected verifier's criterion. In Nightmare mode, verifiers are secretly remapped; the player queries one verifier, but the response corresponds to another verifier's logic, unknown to the player. This mapping must be deduced as part of the reasoning process.

Dataset	Multi-Turn	Multi-Step	No Knowledge	Ground true	Intermediate Eval	Reasoning	Domain
Avalonbench	●	●	●	○	○	●	Game
Multi-LogiEval	○	●	○	●	○	●	Narrative
BoardgameQA	○	●	◖	●	○	◖	Game
MuSR	○	●	○	●	○	●	Narrative
AIME 2024	○	●	●	●	○	●	Math
DSGBench	●	○	○	○	◖	◖	Game
MR-Ben	○	●	●	●	●	●	Science
LOGICGAME	○	●	●	●	○	●	Game
Ours	●	●	●	●	●	●	Game

Table 1: Comparison of multi-round reasoning benchmarks across six key criteria. A ●indicates presence of the feature, a ○means no presence of the feature, and a ◖ indicates partial. The "Domain" column shows the task type of each benchmark.

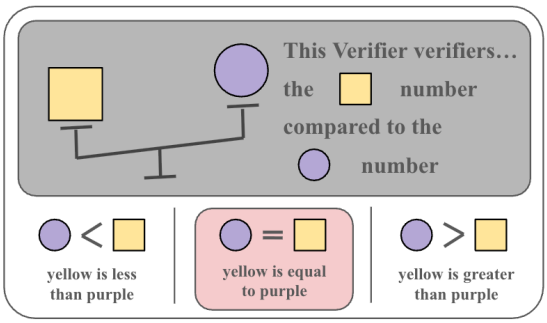

Figure 2: Example of a verifier card used in the deduction game. This verifier compares the values assigned to two colors (in this case, yellow and purple) and returns one of three outcomes: less than, equal to, or greater than. The true decision criterion is highlighted in the middle red box—in this case, verifying whether yellow equals purple.

3.2 TurnBench Construction

3.2.1 Game Setups

Each TurnBench game instance consists of a specific verifier combination, one hidden active rule per verifier, and the unique correct code. For Nightmare mode, each game additionally includes a fixed or dynamically generated hidden mapping between verifiers. We curated 270 Classic and 270 Nightmare game setups (90 per difficulty level), sourced from official materials[2]. All setups are reproducible, and Nightmare mappings are pre-fixed or regenerated at runtime to reduce memorization risk.

3.2.2 Verifier Design

Verifiers are central to TurnBench and encode simple numerical rules (e.g., Figure 2). We incorporate 48 official verifier types, each associated with 2 to 9 potential rules. Since the physical game's verifier logic isn't directly compatible with a simulation environment, we designed a Hidden Condition Selection Algorithm [3] that selects one active rule per verifier to align with the game's design and balance.

3.2.3 LLM Interaction Flow

At game start, the system presents the LLM with the full game context: background, rules, objective, and all verifier definitions. The model then interacts turn-by-turn as described in Section 3.1, adhering to a strict output protocol. In each round:

- In the **Proposal** step, the LLM outputs a code in the format `<CHOICE>: BLUE=X, YELLOW=Y, PURPLE=Z`.
- In the **Verifier Query** step, it selects verifiers with `<CHOICE>: [num]`. Each returns PASS or FAIL.
- In the **Deduce** step, the LLM either submits the code again using the same format or skips the round via `<CHOICE>: SKIP`.
- In Chain-of-Thought (CoT) mode, the LLM also outputs reasoning before decisions using `<REASONING>`.

If the LLM produces malformed output or illegal actions (e.g., invalid verifier ID), a retry mechanism prompts re-generation, while tracking error frequency. Detailed prompts and retry protocols are in the Appendix.

3.2.4 Evaluating Model Reasoning Process

While existing benchmarks focus solely on final answers, TurnBench introduces an automated method for evaluating intermediate reasoning. Specifically, in Classic mode, a model's reasoning involves two phases: (1) inferring each verifier's hidden criterion, and (2) using these to deduce the final code.

[2] https://www.turingmachine.info/

[3] See our repo: `criteria_detector.py`

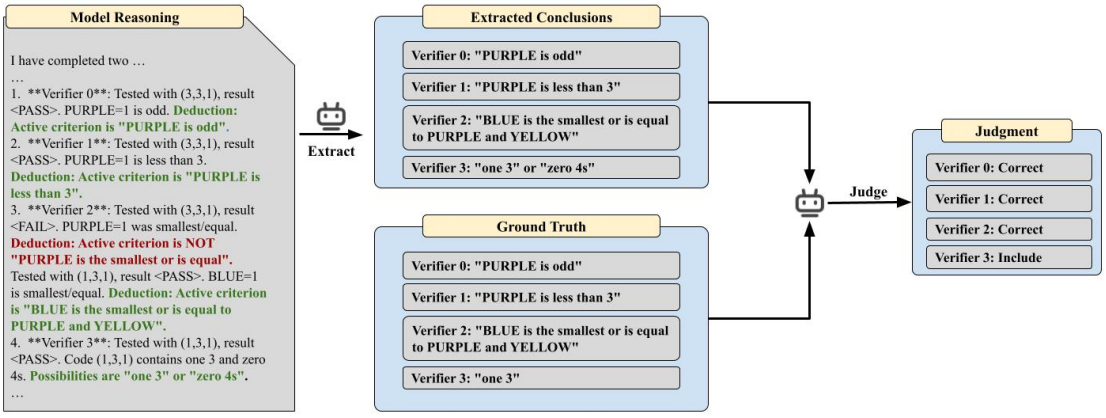

Figure 3: Reasoning Evaluation Pipeline. The LLMs first performs reasoning over verifier results and outputs natural language explanations. From these explanations, active criteria are automatically extracted, then compared against the predefined ground truth verifier logic. The extracted conclusions are judged as Correct, Incorrect, or Include based on semantic agreement with the ground truth.

Since both ground truths (criteria and final code) are known, we can semantically compare model inferences with them.

Our evaluation pipeline (Figure 3) involves two LLM-based components. First, an **Inference Extractor** (Gemini-2.5-Flash (Google, 2025)) parses the model's `<REASONING>` output to identify its explicit claim about a verifier's hidden rule (e.g., "Verifier 1 uses 'BLUE is odd'"). Second, a **Judger**, also Gemini-2.5-Flash, compares the extracted rule to the ground truth and classifies it as: **Correct** (semantically equivalent), **Incorrect** (completely wrong or missing the correct rule), or **Include** (partial overlap with the ground truth).

We validated this automated process through manual inspection. Stratified sampling selected 120 outputs (5% of total), prioritizing failed games for robustness. Manual checks revealed the inference extractor missed 13.7% of applicable conclusions, but achieved 99.7% precision. The Judger reached 99.4% classification accuracy. These results confirm that TurnBench provides a reliable mechanism for process-level evaluation of LLM reasoning.

4 Experiment

4.1 Experiment Setup

To comprehensively explore the limitations of current large language models (LLMs) in multi-turn and multi-step reasoning tasks, we selected both commercial and widely-used open-source models for evaluation, employing different prompting methods. The commercial models include gemini-2.5-flash-preview-04-17 (thinking) (Google, 2025), gpt-o4-mini-high-0416 (thinking) (OpenAI, 2025), and gpt-4.1-2025-04-14. The open-source models tested are deepseek-r1 (thinking) (DeepSeek-AI, 2025), llama-4-maverick (Meta, 2025), mistral-8b (team, 2025), llama-3.1-8b-instruct (AI@Meta, 2024), and qwen-2.5-7b-instruct (Yang et al., 2024). We also evaluated two prompting strategies: Answer Only (AO) and Chain of Thought (CoT) (Wei et al., 2022).

To thoroughly test the reasoning abilities of the state-of-the-art models, all "Thinking" models had their reasoning effort set to "high." Additionally, to compare the reasoning gap between the most advanced LLMs and humans, we invited five human participants with no prior experience with the game to take part in the experiment.

We evaluated two game modes: Classic and Nightmare. Each mode's scenarios were divided equally into three difficulty levels: easy, standard, and hard. For Classic mode, we constructed 270 benchmark scenarios (90 per difficulty). For Nightmare mode, 45 scenarios were selected (15 per difficulty). Human participants played 45 Classic mode games (15 per difficulty), with the Nightmare mode evaluation set matching the models'.

All models and human participants were tested under identical conditions, with the same task prompts and problem setups. To ensure parity in information access, we developed a user interface for humans that displayed exactly the same text as the models saw at each step. Humans were also asked to record their reasoning and thought processes throughout.

Models	Total Avg Acc		Easy Avg Acc		Medium Avg Acc		Hard Avg Acc		Win Avg Turn		Win Avg VER	
	OA	CoT	OA	CoT	OA	CoT	OA	CoT	OA	CoT	OA	CoT
gpt-o4-mini-high (Thinking)	0.578	**0.815**	0.756	**0.933**	0.7	**0.9**	0.278	**0.611**	16	16	7	7
gemini-2.5-flash (Thinking)	**0.652**	0.785	**0.844**	0.9	**0.756**	0.867	**0.356**	0.589	13	**13**	**6**	**6**
deepseek-r1 (Thinking)	0.511	0.63	0.733	0.756	0.511	0.722	0.289	0.411	**12**	**13**	**6**	**6**
gpt-4.1	0.052	0.63	0.078	0.8	0.033	0.689	0.044	0.4	41	15	21	7
llama-4-maverick	0.07	0.326	0.133	0.444	0.056	0.367	0.022	0.167	28	17	12	8
llama-3.1-8b-instruct	0.007	0.015	0.011	0.022	0.011	0	0	0.022	23	13	11	6
mistral-8b	0	0.015	0	0.011	0	0.022	0	0.011	-	8	-	4
qwen-2.5-7b-instruct	0.015	0.022	0.011	0.067	0.022	0	0.011	0	34	6	17	3
Random Guess	0.0082		0.0084		0.008		0.0083		-	-	-	-
Best Human	1		1		1		1		18		8	
Human Average	0.96		0.983		0.947		0.947		20		11	

Table 2: Performance of different models on the Classic Game setting. Metrics include total, easy, medium, and hard average accuracy, as well as average number of turns and average number of verifiers used in successfully won games. Models using Chain-of-Thought (CoT) prompting, especially "Thinking" models like gpt-o4-mini-high and gemini-2.5-flash, consistently outperform standard baselines. Fewer average turns and verifier uses in winning games suggest greater reasoning efficiency. Human and random guess baselines are included for comparison.

Thinking Models	Total Avg Accuracy	Easy Avg Accuracy	Medium Avg Accuracy	Hard Avg Accuracy	Win Avg Turns	Win Avg Verifier Uses
gpt-o4-mini-high	0.111	**0.133**	0.200	0.000	21	8
gemini-2.5-flash	**0.178**	**0.133**	**0.267**	**0.133**	16	8
deepseek-r1	0.067	0.067	0.067	0.067	**12**	**6**
Random Guess	0.0076	0.0074	0.0079	0.0075	-	-
Best Human	1	1	1	1	40	20
Human Average	0.942	0.96	0.933	0.933	38	17

Table 3: Performance of different models on the Nightmare game setting. Metrics include total, easy, medium, and hard average accuracy, as well as win rates measured by average successful turns and verifier-correct final states. Compared to Classic mode, accuracy drops significantly, highlighting the increased difficulty of Nightmare mode. Human players maintain robust performance, while models struggle to generalize under this challenging scenario. Random guess and human baselines are included for comparison.

4.2 Results and Findings

Finding 1: LLMs significantly lag behind humans in multi-turn, multi-step reasoning.

We analyzed overall performance using average accuracy metrics segmented by difficulty (overall, easy, medium, hard), as well as the average number of turns and verifier uses in games won successfully. Fewer turns and verifier uses indicate stronger reasoning ability.

First, we discuss Classic mode results (Table 2). Smaller standard models struggled significantly despite understanding game rules and response format. They had difficulty leveraging verifier feedback to make effective inferences. Because TurnBench requires no external knowledge and relies solely on numerical rules, this suggests that complex reasoning needs models of a certain size and capacity.

Chain of Thought (CoT) prompting consistently improved performance across accuracy metrics and helped "Thinking" models as well. For example, the best-performing gpt-o4-mini-high increased its overall accuracy from 0.578 (AO) to 0.815 (CoT). Larger standard models also showed notable gains, e.g., gpt-4.1 rose from 0.052 (AO) to 0.63 (CoT), and llama-4 from 0.07 to 0.326.

Across difficulty levels, "Thinking" models like gpt-o4-mini-high significantly outperformed standard chat models. For example, the best standard model, gpt-4.1, with CoT only matched the weakest "Thinking" model, DeepSeek, achieving 63% accuracy. This trend was consistent across all difficulties.

CoT prompting also helped models succeed with fewer turns and verifier uses (e.g., gpt-4.1) dropped from 41 to 15 turns and from 21 to 7 verifiers). However, "Thinking" models showed little difference between AO and CoT for turns and verifier use, possibly because they internally perform stepwise reasoning. CoT may mainly help them articulate their reasoning process more clearly and use it as memory for subsequent steps.

Despite improvements, a significant gap remains between LLMs and humans. The "Best Human" achieved 100% accuracy across all metrics, whereas gpt-o4-mini-high (CoT) reached only 81.5%. Humans also outperformed models in average turns (20) and verifier uses (11). Analysis of reasoning logs showed that while models sometimes integrated more clues, humans tended to take more turns (especially on hard tasks) but maintained near-perfect accuracy.

To further test limits, we compared "Thinking" models with CoT against humans in the more challenging Nightmare mode (Table 3). All LLMs' accuracy dropped drastically compared to Classic mode. For example, gpt-o4-mini-high fell from 0.815 to 0.111 overall, and failed completely on Hard difficulty (0 accuracy). The best performing gemini-2.5-flash only reached 0.133. Humans maintained extremely high performance, with the "Best Human" at 100% accuracy and the average human still achieving 94.2%.

These results clearly demonstrate that although "Thinking" models and CoT prompting improve performance, LLMs still lag far behind humans in complex multi-turn, multi-step reasoning tasks, especially under high difficulty. This highlights the substantial gap remaining between current models and human reasoning capabilities.

> **Finding 2:** Once LLMs make a mistake in multi-turn reasoning, they struggle to recover.

In this experiment, we conducted an in-depth analysis of the persistence and evolution of error states in LLMs during multi-turn reasoning. This analysis is based on the thinking process evaluation method described in Section 3.2.4. The results clearly demonstrate that in complex multi-turn reasoning chains, once current LLMs make an initial error, they tend to "lose their way" and struggle to recover autonomously, significantly reducing their final task success rate.

Path Divergence after Initial Errors. Using a Sankey diagram (Figure 4), we tracked model behavior following the First Incorrect Conclusion (FIC). The diagram shows that a large proportion of error paths led directly to "No Subsequent Conclusion," indicating that models often cease reasoning along that path after an initial mistake. Another substantial fraction continued producing incorrect

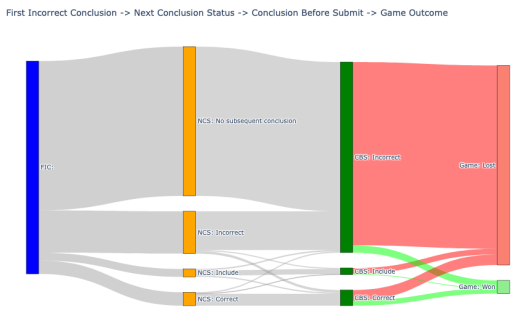

Figure 4: Flow analysis of verifier reasoning paths originating from a First Incorrect Conclusion (FIC). The diagram traces these paths through the Next Conclusion Status (NCS), the verifier's final Conclusion Before Submit (CBS), and the ultimate Game Outcome. A significant proportion of initial errors result in "No subsequent conclusion" or an "Incorrect" NCS, leading to an overwhelmingly "Incorrect" CBS and subsequent "Game Lost" outcomes.

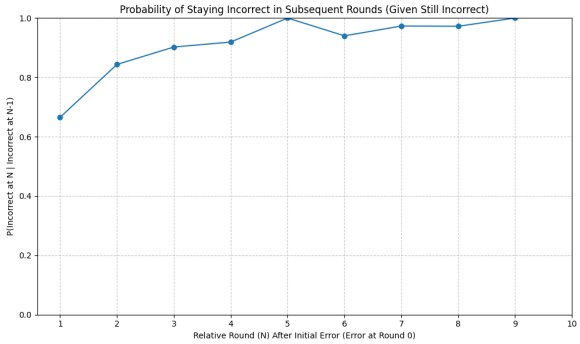

Figure 5: Probability of a model remaining incorrect in each subsequent round after its initial error, conditioned on it being incorrect in the previous round. The likelihood of continuing in an incorrect state increases with each turn, approaching near certainty beyond the fifth round. This trend highlights the models' limited capacity for self-correction once they enter an error state.

conclusions. In contrast, paths that quickly shifted to "Include Correct Components" or "Completely Correct" were relatively rare. Examining how these paths evolved to the Final Conclusion State Before Submission (CBS), we found that those with either "No Subsequent Conclusion" or "Subsequent Incorrect Conclusion" overwhelmingly ended in an incorrect final conclusion. Consequently, these error paths almost always resulted in "Game Lost." Only a small minority of paths that rapidly adjusted to correct or partially correct conclusions after the first error were associated with a higher likelihood of "Game Won." This divergence visually confirms that after the first mistake, models

	llama-3.1-8b	gemini-2.5-flash	gpt-o4-mini-high	gpt-4.1	mistral-8b	llama-4-maverick	qwen-2.5-7b	deepseek-r1
Initial verifier errors	368	96	**66**	141	255	142	318	144
Persistence of initial errors (%)	89.94	91.67	**53.03**	86.52	90.20	63.38	99.06	93.06
Ended with no final conclusion (%)	74.18	71.87	**34.85**	54.61	53.33	45.77	96.23	86.11
Next-turn still incorrect (%)	17.66	19.79	27.27	33.33	38.82	25.35	**3.14**	6.94
Success despite persistent errors (%)	1.08	12.72	**32.14**	13.41	0.66	8.11	0.54	7.87
Success when no / fixed errors (%)	1.75	**95.34**	87.55	84.57	3.13	41.75	8.00	90.56

Table 4: Comparison of large language models on their ability to handle verifier errors during multi-turn reasoning. Metrics include the number of initial verifier errors, error persistence rate, likelihood of remaining incorrect in the next turn, and task success rates depending on error persistence or correction. The table highlights key differences in robustness and recovery among models such as GPT-4.1, LLaMA-3.1-8B, and Gemini-2.5-Flash.

rarely self-correct and tend either to halt reasoning or perpetuate errors—an initial indication of "losing their way."

Solidification and Persistence of Error States. To further investigate error dynamics, we analyzed model behavior after making an error. Error states proved extremely "sticky." Figure 5 depicts the probability that a model continues to produce incorrect conclusions in subsequent relative rounds, given that it is currently incorrect. In the first relative round after the initial error (X=1), if the model outputs a conclusion, there is already approximately a 65–70% chance it is incorrect. Alarmingly, this probability rises sharply with additional rounds, nearing 100% by the fifth relative round. This suggests that once a model enters several consecutive rounds of incorrect reasoning, it almost completely loses the ability to break the error cycle.

5 Conclusion

In this paper, our investigation using TurnBench has clarified the capabilities and limitations of Large Language Models (LLMs) in multi-turn, multi-step reasoning. TurnBench addresses several key limitations of current benchmarks and offers an effective method for automatically analyzing the reasoning processes of LLMs. Using this framework, we evaluated multiple standard chat models and thinking models, uncovering key findings that highlight the limitations of existing models. In summary, TurnBench fills a gap in the evaluation of LLMs' multi-turn, multi-step reasoning capabilities and provides a novel solution for assessing model reasoning processes. We hope that our work will inspire further research into multi-turn reasoning.

Limitation

Effectively and accurately measuring a model's thinking process has always been a challenge. The automated evaluation of model thinking processes proposed in this paper requires an evaluation framework built on rules, which lacks generality. Furthermore, using Gemini 2.5 Flash for model inference extraction still has certain limitations. Although the extracted results have shown high accuracy after manual evaluation, further research and optimization are still needed.

References

AI@Meta. 2024. Llama 3 model card.

Elif Akata, Lion Schulz, Julian Coda-Forno, Seong Joon Oh, Matthias Bethge, and Eric Schulz. 2023. Playing repeated games with large language models. *arXiv preprint arXiv:2305.16867*.

AoPS. 2024. Aime 2024. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Debrup Das, Debopriyo Banerjee, Somak Aditya, and Ashish Kulkarni. 2024. Mathsensei: a tool-augmented large language model for mathematical reasoning. *arXiv preprint arXiv:2402.17231*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Kevin N. Dunbar and David Klahr. 2012. 701 scientific thinking and reasoning. In *The Oxford Handbook of Thinking and Reasoning*. Oxford University Press.

Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. 2022. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074.

Google. 2025. Gemini 2.5 flash preview. *https://storage.googleapis.com/model-cards/documents/gemini-2.5-flash-preview.pdf*.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, et al. 2024. Llm

reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*.

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

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Jonathan Light, Min Cai, Sheng Shen, and Ziniu Hu. 2023. Avalonbench: Evaluating llms playing the game of avalon. *arXiv preprint arXiv:2310.05036*.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. Criticbench: Benchmarking llms for critique-correct reasoning. *arXiv preprint arXiv:2402.14809*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Meta. 2025. llama models. https://github.com/meta-llama/llama-models.

Theo X Olausson, Alex Gu, Benjamin Lipkin, Cedegao E Zhang, Armando Solar-Lezama, Joshua B Tenenbaum, and Roger Levy. 2023. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. *arXiv preprint arXiv:2310.15164*.

OpenAI. 2025. Openai o3 and o4-mini system card. *https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf*.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. 2025. Balrog: Benchmarking agentic llm and vlm reasoning on games.

Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. 2024. Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models. *arXiv preprint arXiv:2406.17169*.

John Schultz, Jakub Adamek, Matej Jusup, Marc Lanctot, Michael Kaisers, Sarah Perrin, Daniel Hennes, Jeremy Shar, Cannada Lewis, Anian Ruoss, et al. 2024. Mastering board games by external and internal planning with language models. *arXiv preprint arXiv:2412.12119*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. 2023. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. *arXiv preprint arXiv:2310.16049*.

Wenjie Tang, Yuan Zhou, Erqiang Xu, Keyan Cheng, Minne Li, and Liquan Xiao. 2025. Dsgbench: A diverse strategic game benchmark for evaluating llm-based agents in complex decision-making environments. *arXiv preprint arXiv:2503.06047*.

Mistral AI team. 2025. Un ministral, des ministraux. https://mistral.ai/news/ministraux.

Nemika Tyagi, Mihir Parmar, Mohith Kulkarni, Aswin Rrv, Nisarg Patel, Mutsumi Nakamura, Arindam Mitra, and Chitta Baral. 2024. Step-by-step reasoning to solve grid puzzles: Where do llms falter? *arXiv preprint arXiv:2407.14790*.

Boshi Wang, Xiang Yue, and Huan Sun. 2023a. Can chatgpt defend its belief in truth? evaluating llm reasoning via debate. *arXiv preprint arXiv:2305.13160*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Peter Cathcart Wason and Philip Nicholas Johnson-Laird. 1972. *Psychology of Reasoning: Structure and Content*. Harvard University Press, Cambridge, MA, USA.

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

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. 2022. Naturalprover: Grounded mathematical proof generation with language models. *Advances in Neural Information Processing Systems*, 35:4913–4927.

Bryan Wilie, Samuel Cahyawijaya, Etsuko Ishii, Junxian He, and Pascale Fung. 2024. Belief revision: The adaptability of large language models reasoning. *arXiv preprint arXiv:2406.19764*.

Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu. 2023. Exploring large language models for communication games: An empirical study on werewolf. *arXiv preprint arXiv:2309.04658*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Zhongshen Zeng, Yinhong Liu, Yingjia Wan, Jingyao Li, Pengguang Chen, Jianbo Dai, Yuxuan Yao, Rongwu Xu, Zehan Qi, Wanru Zhao, et al. 2024. Mr-ben: A meta-reasoning benchmark for evaluating system-2 thinking in llms. *arXiv preprint arXiv:2406.13975*.

Ruiwen Zhou, Wenyue Hua, Liangming Pan, Sitao Cheng, Xiaobao Wu, En Yu, and William Yang Wang. 2024. Rulearena: A benchmark for rule-guided reasoning with llms in real-world scenarios. *arXiv preprint arXiv:2412.08972*.

Richard Zhuang, Akshat Gupta, Richard Yang, Aniket Rahane, Zhengyu Li, and Gopala Anumanchipalli. 2025. Pokerbench: Training large language models to become professional poker players. *arXiv preprint arXiv:2501.08328*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.

A Prompts Used in Experiments

Classic – OA Prompt

```
classic system_prompt = """You are participating in a competitive logic deduction
↪  game called Turing Machine.
Your goal is to win first place by deducing a secret 3-digit code with minimal
↪  rounds and verifier usage, but accuracy takes priority over speed.

Game Objective:
- Deduce the secret 3-digit code made up of digits 1-5.
- BLUE = first digit, YELLOW = second digit, PURPLE = third digit.
- Each digit can be 1, 2, 3, 4, or 5 (digits may repeat).
- The code is the ONLY combination that satisfies the active criterion of ALL
↪  chosen verifiers.

Game Structure (Rounds):
1. Proposal: Design a 3-digit code to test (format: BLUE=X, YELLOW=Y, PURPLE=Z,
↪  where X, Y, Z are digits from 1 to 5).
2. Question: Sequentially choose 0 to 3 verifiers to test your proposed code each
↪  round. After each selection, you will see the result, and then you can decide
↪  whether to select the next one.
3. Deduce: Based on verifier results, you can submit a final answer or continue
↪  to the next round.
4. End Round: If you didn't submit a final answer, a new round begins.

Verifier Rules:
- Each verifier checks ONE specific property (criterion) about the code.
- Each verifier has multiple potential criteria, but for each game, only ONE is
↪  secretly selected as 'active'. You don't know which criterion is active for
↪  any given verifier.
- Focus of Verification: When testing your code against a verifier, it
↪  exclusively evaluates it against its single, active criterion. The verifier
↪  completely ignores all other potential criteria, including its own inactive
↪  ones.
- PASS Condition: A verifier returns `<PASS>` if and only if your code satisfies
↪  this single active criterion.
- FAIL Condition: A verifier returns `<FAIL>` if and only if your code does not
↪  satisfy this single active criterion.
- Non-Overlapping Information: The active criteria selected across different
↪  verifiers for a game will provide distinct information.

Winning Strategy:
- It is possible to deduce the solution through joint reasoning, utilizing the
↪  combined results of multiple verifiers along with system rules such as the
↪  existence of a unique solution and the principle that no two verifiers offer
↪  redundant information.
- Only submit a final guess when you have either tested all verifiers and
↪  received <PASS> for each, or when your reasoning clearly proves your code
↪  satisfies all possible active verifier criteria. Accuracy takes priority over
↪  speed.

Current Game Setup:
{game_setup}
"""

classic_proposal_prompt_with_hint = """You are now entering the **Proposal
↪  Stage** of this round.

**Stage Purpose**:
In this stage, you need to compose a 3-digit code to help you to gather
↪  information from the verifiers. The code can NOT be changed in the subsequent
↪  stages of this round.

**3-digit code rules**:
- BLUE = first digit (X), YELLOW = second digit (Y), PURPLE = third digit (Z).
- Each digit (X, Y, Z) can be 1, 2, 3, 4, or 5. Digits may repeat.

**Your Goal in This Stage**:
- Design a code that will test a specific hypothesis.
```

```
- Think about what a <PASS> or <FAIL> would tell you.
- Choose a code that lets you learn something meaningful from verifiers.

**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <CHOICE>: BLUE=1, YELLOW=1, PURPLE=1
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: BLUE=X, YELLOW=Y, PURPLE=Z
"""

classic_not_valid_proposal_format_prompt_with_hint = """You did not follow the
↪  required response format. Please try again with same code.

**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <PROPOSAL>: BLUE=1, YELLOW=1, PURPLE=1
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: BLUE=[X], YELLOW=[Y], PURPLE=[Z]
"""

classic_first_question_prompt_with_hint = """You are now entering the **Verifier
↪  Questioning Stage** of this round.

**Current Verifiers**:
{verifier_descriptions}

**Stage Purpose**:
In this stage, you can test your proposed 3-digit code using verifiers. Each
↪  verifier checks one hidden criterion. Use the test results to gather
↪  information and refine your deduction.

**Verifier Rules Summary**:
- Each verifier has ONE secretly selected active criterion.
- <PASS> means your code satisfies this rule; <FAIL> means it does not.
- Active rules do NOT overlap between verifiers.

**Your Goal in This Stage**:
- Choosing verifiers is optional; testing 0 verifiers is allowed. If you want to
↪  choose the verifier, you must choose verifiers **one at a time**. After each
↪  result, you may decide whether to test another. You may choose to test 0 to 3
↪  verifiers **in total** during this round.
- **Passing all tested verifiers does NOT mean the code is correct.** To win,
↪  your code must satisfy the hidden criterion of **all verifiers**, whether
↪  tested or not.

**What You Must Do Now**:
- If you want to choose a verifier to test your proposed code, reply with
↪  verifier_num after <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

classic_following_question_prompt_with_hint = """You chose Verifier
↪  <{verifier_num}> and the result is <{verifier_result}>.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.
```

```
**Response format**:
<CHOICE>: [your_choice]
"""

classic_after_last_question_prompt_with_hint = """You chose Verifier
↪  <{verifier_num}> and the result is <{verifier_result}>.

You have now tested the maximum number of three verifiers for this round. The
↪  next stage is the Deduce Stage. If you want to test more verifiers or new
↪  code, you can choose SKIP during the Deduce Stage to move on to the next
↪  round.
"""

classic_not_valid_question_format_prompt_with_hint = """You did not follow the
↪  required response format. Please try again with same choice.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

classic_not_valid_verifier_choice_prompt_with_hint = """You selected Verifier
↪  <{verifier_num}>, which is not a valid verifier number.

Please choose a valid verifier or SKIP to next stage.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

classic_deduce_prompt_with_hint = """You are now entering the **Deduce Stage** of
↪  this round.

**Stage Purpose**:
In this stage, you can analyze all the information gathered then decide whether
↪  to continue to the next round or submit a final guess.

**Hint**:
- Passing all tested verifiers does not mean the code is correct if not all
↪  verifiers were tested. To be correct, the code must satisfy the hidden
↪  criteria of all verifiers, not just the ones you tested.
- You may choose not to test some verifiers if you can clearly reason that your
↪  code meets their requirements. But you must ensure every verifier is either
↪  tested and passed, or clearly justified through reasoning. Testing and
↪  passing only part of the verifiers is not enough if others are ignored.
- This stage **is not for testing**, you don't have to submit an answer; you can
↪  proceed to the next round to continue gathering information.
- Accuracy takes priority over speed. If you submit, the game will end, and an
↪  incorrect guess will result in immediate failure.

**Your Goal in This Stage**:
- Decide whether to submit the final guess or continue to the next round. Submit
↪  the final guess will end the game, continue to the next round will help you
↪  gather more information.
- Submission is not mandatory, you must make this decision based on your own
↪  reasoning.
```

```
**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

classic_deduce_result_prompt_with_hint = "The final guess is {submitted_code}.
↪  The answer is {answer}, the guess is {is_correct}."

classic_not_valid_deduce_format_prompt_with_hint = """You did NOT follow the
↪  response format. Please try again.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""
```

Nightmare – OA Prompt

```
nightmare_system_prompt = """You are participating in a competitive logic
↪  deduction game called Turing Machine.
Your goal is to win first place by deducing a secret 3-digit code with minimal
↪  rounds and verifier usage, but accuracy takes priority over speed.

Game Objective:
- Deduce the secret 3-digit code made up of digits 1-5.
- BLUE = first digit, YELLOW = second digit, PURPLE = third digit.
- Each digit can be 1, 2, 3, 4, or 5 (digits may repeat).
- The code is the ONLY combination that satisfies the active criterion of ALL
↪  chosen verifiers.

Game Structure (Rounds):
1. Proposal: Design a 3-digit code to test (format: BLUE=X, YELLOW=Y, PURPLE=Z,
↪  where X, Y, Z are digits from 1 to 5).
2. Question: Sequentially choose 0 to 3 verifiers to test your proposed code each
↪  round. After each selection, you will see the result from an unknown
↪  verifier. The verifier identity will be hidden.
3. Deduce: Based on verifier results, you can submit a final answer or continue
↪  to the next round.
4. End Round: If you didn't submit a final answer, a new round begins.

Verifier Rules:
- Each verifier checks ONE specific property (criterion) about the code.
- Each verifier has multiple potential criteria, but for each game, only ONE is
↪  secretly selected as 'active'. You don't know which criterion is active for
↪  any given verifier.
- Focus of Verification: When testing your code against a verifier, it
↪  EXCLUSIVELY evaluates it against its SINGLE, ACTIVE criterion. The verifier
↪  completely ignores all other potential criteria, including its own inactive
↪  ones.
- In this game, you don’t know which Verifier’s result you’re actually seeing --
↪  the mapping between Verifiers and their displayed results is randomized and
↪  hidden from the player, though fixed for the entire game.
- PASS Condition: A verifier returns `<PASS>` if and only if your code satisfies
↪  the active criterion of the actual Verifier it is mapped to. For example, if
↪  Verifier 1 is secretly mapped to Verifier 2, then a <PASS> from Verifier 1
↪  means your code met Verifier 2's hidden active rule.
```

```
- FAIL Condition: A verifier returns `<FAIL>` if and only if your code does not
↪  satisfy the active criterion of the actual Verifier it is mapped to. A <FAIL>
↪  simply means the mapped Verifier's rule was not met.
- Non-Overlapping Information: The active criteria selected across different
↪  verifiers for a game will provide distinct information.

Winning Strategy:
- It is possible to deduce the solution through joint reasoning, utilizing the
↪  combined results of multiple verifiers along with system rules such as the
↪  existence of a unique solution and the principle that no two verifiers offer
↪  redundant information.
- One possible strategy is to carefully modify your code across multiple rounds
↪  and observe how each Verifier’s output changes. By analyzing the pattern of
↪  responses, you can infer the hidden mapping between Verifiers and their
↪  actual criteria.
- Only submit a final guess when you have either tested all verifiers and
↪  received <PASS> for each, or when your reasoning clearly proves your code
↪  satisfies all possible active verifier criteria. Accuracy takes priority over
↪  speed.

Current Game Setup:
{game_setup}
"""

nightmare_proposal_prompt_with_hint = """You are now entering the **Proposal
↪  Stage** of this round.

**Stage Purpose**:
In this stage, you need to compose a 3-digit code to help you to gather
↪  information from the verifiers. The code can NOT be changed in the subsequent
↪  stages of this round.

**3-digit code rules**:
- BLUE = first digit (X), YELLOW = second digit (Y), PURPLE = third digit (Z).
- Each digit (X, Y, Z) can be 1, 2, 3, 4, or 5. Digits may repeat.

**Your Goal in This Stage**:
- Design a code that will test a specific hypothesis.
- Think about what a <PASS> or <FAIL> would tell you, but you don’t know which
↪  Verifier’s result you’re actually seeing -- the mapping between Verifiers and
↪  their displayed results is randomized and hidden from the player, though
↪  fixed for the entire game.
- Choose a code that lets you learn something meaningful from verifiers.

**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <CHOICE>: BLUE=1, YELLOW=1, PURPLE=1
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: BLUE=X, YELLOW=Y, PURPLE=Z
"""

nightmare_first_question_prompt_with_hint = """You are now entering the
↪  **Verifier Questioning Stage** of this round.

**Current Verifiers**:
{verifier_descriptions}

**Stage Purpose**:
In this stage, you can test your proposed 3-digit code using verifiers. Each
↪  verifier checks one hidden criterion. Use the test results to gather
↪  information and refine your deduction.

**Verifier Rules Summary**:
- Each verifier has ONE secretly selected active criterion.
- Each verifier shows results for a different, hidden verifier (the mapping is
↪  randomized but fixed for the entire game).
- <PASS> means your code satisfies the active criterion of the secretly mapped
↪  verifier. <FAIL> means your code does not satisfy that criterion.
```

```
- Active rules do NOT overlap between verifiers.

**Your Goal in This Stage**:
- Choosing verifiers is optional; testing 0 verifiers is allowed. If you want to
↪  choose the verifier, you must choose verifiers **one at a time**. After each
↪  result, you may decide whether to test another. You may choose to test 0 to 3
↪  verifiers **in total** during this round.
- **Passing all tested verifiers does NOT mean the code is correct.** To win,
↪  your code must satisfy the hidden criterion of **all verifiers**, whether
↪  tested or not.

**What You Must Do Now**:
- If you want to choose a verifier to test your proposed code, reply with
↪  verifier_num after <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_following_question_prompt_with_hint = """You chose Verifier
↪  <{verifier_num}> and the result is <{verifier_result}>.

**Hint**:
- `<PASS>` means your code satisfies the active criterion of the actual Verifier
↪  it is mapped to. For example, if Verifier 1 is secretly mapped to Verifier 2,
↪  then a <PASS> from Verifier 1 means your code met Verifier 2's hidden active
↪  rule.
- `<FAIL>` means your code does not satisfy the active criterion of the actual
↪  Verifier it is mapped to.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_after_last_question_prompt_with_hint = """You chose Verifier
↪  <{verifier_num}> and the result is <{verifier_result}>.

You have now tested the maximum number of three verifiers for this round. The
↪  next stage is the Deduce Stage. If you want to test more verifiers or new
↪  code, you can choose SKIP during the Deduce Stage to move on to the next
↪  round.
"""

nightmare_not_valid_question_format_prompt_with_hint = """You did not follow the
↪  required response format. Please try again with same choice.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_not_valid_verifier_choice_prompt_with_hint = """You selected Verifier
↪  <{verifier_num}>, which is not a valid verifier number.
```

```
Please choose a valid verifier or SKIP to next stage.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_deduce_prompt_with_hint = """You are now entering the **Deduce Stage**
↪  of this round.

**Stage Purpose**:
In this stage, you can analyze all the information gathered then decide whether
↪  to continue to the next round or submit a final guess.

**Hint**:
- Passing all tested verifiers does not mean the code is correct if not all
↪  verifiers were tested. To be correct, the code must satisfy the hidden
↪  criteria of all verifiers, not just the ones you tested.
- You may choose not to test some verifiers if you can clearly reason that your
↪  code meets their requirements. But you must ensure every verifier is either
↪  tested and passed, or clearly justified through reasoning. Testing and
↪  passing only part of the verifiers is not enough if others are ignored.
- This stage **is not for testing**, you don't have to submit an answer; you can
↪  proceed to the next round to continue gathering information.
- Accuracy takes priority over speed. If you submit, the game will end, and an
↪  incorrect guess will result in immediate failure.

**Your Goal in This Stage**:
- Decide whether to submit the final guess or continue to the next round. Submit
↪  the final guess will end the game, continue to the next round will help you
↪  gather more information.
- Submission is not mandatory, you must make this decision based on your own
↪  reasoning.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_deduce_result_prompt_with_hint = "The final guess is {submitted_code}.
↪  The answer is {answer}, the guess is {is_correct}."

nightmare_not_valid_deduce_format_prompt_with_hint =  """You did NOT follow the
↪  response format. Please try again.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""
```

Classic – CoT Prompt

```
classic_system_prompt = """You are participating in a competitive logic deduction
↪  game called Turing Machine.
Your goal is to win first place by deducing a secret 3-digit code with minimal
↪  rounds and verifier usage, but accuracy takes priority over speed.

Game Objective:
- Deduce the secret 3-digit code made up of digits 1-5.
- BLUE = first digit, YELLOW = second digit, PURPLE = third digit.
- Each digit can be 1, 2, 3, 4, or 5 (digits may repeat).
- The code is the ONLY combination that satisfies the active criterion of ALL
↪  chosen verifiers.

Game Structure (Rounds):
1. Proposal: Design a 3-digit code to test (format: BLUE=X, YELLOW=Y, PURPLE=Z,
↪  where X, Y, Z are digits from 1 to 5).
2. Question: Sequentially choose 0 to 3 verifiers to test your proposed code each
↪  round. After each selection, you will see the result, and then you can decide
↪  whether to select the next one.
3. Deduce: Based on verifier results, you can submit a final answer or continue
↪  to the next round.
4. End Round: If you didn't submit a final answer, a new round begins.

Verifier Rules:
- Each verifier checks ONE specific property (criterion) about the code.
- Each verifier has multiple potential criteria, but for each game, only ONE is
↪  secretly selected as 'active'. You don't know which criterion is active for
↪  any given verifier.
- Focus of Verification: When testing your code against a verifier, it
↪  **exclusively** evaluates it against its **single, active criterion**. The
↪  verifier completely ignores all other potential criteria, including its own
↪  inactive ones.
- PASS Condition: A verifier returns `<PASS>` if and only if your code satisfies
↪  this single active criterion. A `<PASS>` confirms *only* that this specific
↪  rule was met by the tested code.
- FAIL Condition: A verifier returns `<FAIL>` **if and only if** your code does
↪  not satisfy this single active criterion. A `<FAIL>` indicates *only* that
↪  this specific rule was violated by the tested code.
- Non-Overlapping Information: The active criteria selected across different
↪  verifiers for a game will provide distinct information.


Winning Strategy:
- It is possible to deduce the solution through joint reasoning, utilizing the
↪  combined results of multiple verifiers along with system rules such as the
↪  existence of a unique solution and the principle that no two verifiers offer
↪  redundant information.
- Only submit a final guess when you have either tested all verifiers and
↪  received <PASS> for each, or when your reasoning clearly proves your code
↪  satisfies all possible active verifier criteria. Accuracy takes priority over
↪  speed.

Current Game Setup:
{game_setup}
"""

classic_proposal_prompt_with_reasoning_with_hint = """You are now entering the
↪  **Proposal Stage** of this round.

**Stage Purpose**:
In this stage, you need to compose a 3-digit code to help you to gather
↪  information from the verifiers. The code can NOT be changed in the subsequent
↪  stages of this round.

**3-digit code rules**:
- BLUE = first digit (X), YELLOW = second digit (Y), PURPLE = third digit (Z).
- Each digit (X, Y, Z) can be 1, 2, 3, 4, or 5. Digits may repeat.
```

```
**Your Goal in This Stage**:
- Design a code that will test a specific hypothesis.
- Think about what a <PASS> or <FAIL> would tell you.
- Choose a code that lets you learn something meaningful from verifiers.

**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <PROPOSAL>: BLUE=1, YELLOW=1, PURPLE=1
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  code.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing this code]
<CHOICE>: BLUE=[X], YELLOW=[Y], PURPLE=[Z]
"""

classic_not_valid_proposal_format_prompt_with_reasoning_with_hint = """You did
↪  not follow the required response format. Please try again with same code.

**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <PROPOSAL>: BLUE=1, YELLOW=1, PURPLE=1
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  code.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing this code]
<CHOICE>: BLUE=[X], YELLOW=[Y], PURPLE=[Z]
"""

classic_first_question_prompt_with_reasoning_with_hint = """You are now entering
↪  the **Verifier Questioning Stage** of this round.

Current Verifiers:
{verifier_descriptions}

**Stage Purpose**:
In this stage, you can test your proposed 3-digit code using verifiers. Each
↪  verifier checks one hidden criterion. Use the test results to gather
↪  information and refine your deduction.

**Verifier Rules Summary**:
- Each verifier has ONE secretly selected active criterion.
- <PASS> means your code satisfies this rule; <FAIL> means it does not.
- Active rules do NOT overlap between verifiers.

**Your Goal in This Stage**:
- Choosing verifiers is optional; testing 0 verifiers is allowed. If you want to
↪  choose the verifier, you must choose verifiers **one at a time**. After each
↪  result, you may decide whether to test another. You may choose to test 0 to 3
↪  verifiers **in total** during this round.
- **Passing all tested verifiers does NOT mean the code is correct.** To win,
↪  your code must satisfy the hidden criterion of **all verifiers**, whether
↪  tested or not.

**What You Must Do Now**:
- If you want to choose a verifier to test your proposed code, reply with
↪  verifier_num after <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  choice.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing the verifier or
↪  skipping verifiers]
<CHOICE>: [your_choice]
"""
```

```
classic_following_question_prompt_with_reasoning_with_hint = """You chose
↪  Verifier <{verifier_num}> and the result is <{verifier_result}>.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- Explain your reasoning step by step based on verifier result after <REASONING>
↪  tag, then provide your choice.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing the verifier or
↪  skipping verifiers]
<CHOICE>: [your_choice]
"""

classic_after_last_question_prompt_with_reasoning_with_hint = """You chose
↪  Verifier <{verifier_num}> and the result is <{verifier_result}>.

You have now tested the maximum number of three verifiers for this round. The
↪  next stage is the Deduce Stage. If you want to test more verifiers or new
↪  code, you can choose SKIP during the Deduce Stage to move on to the next
↪  round.
"""

classic_not_valid_question_format_prompt_with_reasoning_with_hint = """You did
↪  not follow the required response format. Please try again with same choice.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- Explain your reasoning step by step based on verifier result after <REASONING>
↪  tag, then provide your choice.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing the verifier or
↪  skipping verifiers]
<CHOICE>: [your_choice]
"""

classic_not_valid_verifier_choice_prompt_with_reasoning_with_hint = """You
↪  selected Verifier <{verifier_num}>, which is not a valid verifier number.

Please choose a valid verifier or SKIP to next stage.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- Explain your reasoning step by step based on verifier result after <REASONING>
↪  tag, then provide your choice.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing the verifier or
↪  skipping verifiers]
<CHOICE>: [your_choice]
"""

classic_deduce_prompt_with_reasoning_with_hint = """You are now entering the
↪  **Deduce Stage** of this round.

**Stage Purpose**:
In this stage, you can analyze all the information gathered then decide whether
↪  to submit a final guess or continue to the next round.

**Hint**:
```

```
- Passing all tested verifiers does not mean the code is correct if not all
↪  verifiers were tested. To be correct, the code must satisfy the hidden
↪  criteria of all verifiers, not just the ones you tested.
- You may choose not to test some verifiers if you can clearly reason that your
↪  code meets their requirements. But you must ensure every verifier is either
↪  tested and passed, or clearly justified through reasoning. Testing and
↪  passing only part of the verifiers is not enough if others are ignored.
- This stage **is not for testing**, you don't have to submit an answer; you can
↪  proceed to the next round to continue gathering information.
- Accuracy takes priority over speed. If you submit, the game will end, and an
↪  incorrect guess will result in immediate failure.

**Your Goal in This Stage**:
- Analysis all information gathered.
- Decide whether to submit the final guess or continue to the next round.
- Submission is not mandatory, you must make this decision based on your own
↪  reasoning.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  choice. If you want to submit a final guess, you must provide the reasons for
↪  not proceeding to the next round.

**Response format**:
<REASONING>: [Analysis and explain your reasoning step by step for continue to
↪  next round or submit final guess]
<CHOICE>: [your_choice]
"""

classic_deduce_result_prompt_with_reasoning_with_hint = "The final guess is
↪  {submitted_code}. The answer is {answer}, the guess is {is_correct}."

classic_not_valid_deduce_format_prompt_with_reasoning_with_hint = """You did NOT
↪  follow the response format. Please try again.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  choice.

**Response format**:
<REASONING>: [Analysis and explain your reasoning step by step for submitting the
↪  final guess or continue to next round]
<CHOICE>: [your_choice]
"""
```

Nightmare – CoT Prompt

```
nightmare_system_prompt = """You are participating in a competitive logic
↪  deduction game called Turing Machine.
Your goal is to win first place by deducing a secret 3-digit code with minimal
↪  rounds and verifier usage, but accuracy takes priority over speed.

Game Objective:
- Deduce the secret 3-digit code made up of digits 1-5.
- BLUE = first digit, YELLOW = second digit, PURPLE = third digit.
- Each digit can be 1, 2, 3, 4, or 5 (digits may repeat).
- The code is the ONLY combination that satisfies the active criterion of ALL
↪  chosen verifiers.
```

```
Game Structure (Rounds):
1. Proposal: Design a 3-digit code to test (format: BLUE=X, YELLOW=Y, PURPLE=Z,
↪  where X, Y, Z are digits from 1 to 5).
2. Question: Sequentially choose 0 to 3 verifiers to test your proposed code each
↪  round. After each selection, you will see the result from an unknown
↪  verifier. The verifier identity will be hidden.
3. Deduce: Based on verifier results, you can submit a final answer or continue
↪  to the next round.
4. End Round: If you didn't submit a final answer, a new round begins.

Verifier Rules:
- Each verifier checks ONE specific property (criterion) about the code.
- Each verifier has multiple potential criteria, but for each game, only ONE is
↪  secretly selected as 'active'. You don't know which criterion is active for
↪  any given verifier.
- Focus of Verification: When testing your code against a verifier, it
↪  EXCLUSIVELY evaluates it against its SINGLE, ACTIVE criterion. The verifier
↪  completely ignores all other potential criteria, including its own inactive
↪  ones.
- In this game, you don’t know which Verifier’s result you’re actually seeing --
↪  the mapping between Verifiers and their displayed results is randomized and
↪  hidden from the player, though fixed for the entire game.
- PASS Condition: A verifier returns `<PASS>` if and only if your code satisfies
↪  the active criterion of the actual Verifier it is mapped to. For example, if
↪  Verifier 1 is secretly mapped to Verifier 2, then a <PASS> from Verifier 1
↪  means your code met Verifier 2's hidden active rule.
- FAIL Condition: A verifier returns `<FAIL>` if and only if your code does not
↪  satisfy the active criterion of the actual Verifier it is mapped to. A <FAIL>
↪  simply means the mapped Verifier's rule was not met.
- Non-Overlapping Information: The active criteria selected across different
↪  verifiers for a game will provide distinct information.

Winning Strategy:
- It is possible to deduce the solution through joint reasoning, utilizing the
↪  combined results of multiple verifiers along with system rules such as the
↪  existence of a unique solution and the principle that no two verifiers offer
↪  redundant information.
- One possible strategy is to carefully modify your code across multiple rounds
↪  and observe how each Verifier’s output changes. By analyzing the pattern of
↪  responses, you can infer the hidden mapping between Verifiers and their
↪  actual criteria.
- Only submit a final guess when you have either tested all verifiers and
↪  received <PASS> for each, or when your reasoning clearly proves your code
↪  satisfies all possible active verifier criteria. Accuracy takes priority over
↪  speed.

Current Game Setup:
{game_setup}
"""

nightmare_proposal_prompt_with_reasoning_with_hint = """You are now entering the
↪  **Proposal Stage** of this round.

**Stage Purpose**:
In this stage, you need to compose a 3-digit code to help you to gather
↪  information from the verifiers. The code cannot be changed in the subsequent
↪  stages of this round.

**3-digit code rules**:
- BLUE = first digit (X), YELLOW = second digit (Y), PURPLE = third digit (Z).
- Each digit (X, Y, Z) can be 1, 2, 3, 4, or 5. Digits may repeat.

**Your Goal in This Stage**:
- Design a code that will test a specific hypothesis.
- Think about what a <PASS> or <FAIL> would tell you, but you don’t know which
↪  Verifier’s result you’re actually seeing -- the mapping between Verifiers and
↪  their displayed results is randomized and hidden from the player, though
↪  fixed for the entire game.
- Choose a code that lets you learn something meaningful from verifiers.
```

```
**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <PROPOSAL>: BLUE=1, YELLOW=1, PURPLE=1
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  code.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing this code]
<CHOICE>: BLUE=[X], YELLOW=[Y], PURPLE=[Z]
"""

nightmare_not_valid_proposal_format_prompt_with_reasoning_with_hint = """You did
↪  not follow the required response format. Please try again with same code.

**What You Must Do Now**:
- Reply the code you want to use in this round with required response format. For
↪  example, <PROPOSAL>: BLUE=1, YELLOW=1, PURPLE=1
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  code.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing this code]
<CHOICE>: BLUE=[X], YELLOW=[Y], PURPLE=[Z]
"""

nightmare_first_question_prompt_with_reasoning_with_hint = """You are now
↪  entering the **Verifier Questioning Stage** of this round.

**Current Verifiers**:
{verifier_descriptions}

**Stage Purpose**:
In this stage, you can test your proposed 3-digit code using verifiers. Each
↪  verifier checks one hidden criterion. Use the test results to gather
↪  information and refine your deduction.

**Verifier Rules Summary**:
- Each verifier has ONE secretly selected active criterion.
- Each verifier shows results for a different, hidden verifier (the mapping is
↪  randomized but fixed for the entire game).
- <PASS> means your code satisfies the active criterion of the secretly mapped
↪  verifier. <FAIL> means your code does not satisfy that criterion.
- Active rules do NOT overlap between verifiers.

**Your Goal in This Stage**:
- Choosing verifiers is optional; testing 0 verifiers is allowed. If you want to
↪  choose the verifier, you must choose verifiers **one at a time**. After each
↪  result, you may decide whether to test another. You may choose to test 0 to 3
↪  verifiers **in total** during this round.
- **Passing all tested verifiers does NOT mean the code is correct.** To win,
↪  your code must satisfy the hidden criterion of **all verifiers**, whether
↪  tested or not.

**What You Must Do Now**:
- If you want to choose a verifier to test your proposed code, reply with
↪  verifier_num after <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- Explain your reasoning step by step based on verifier result after <REASONING>
↪  tag, then provide your choice.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing the verifier or
↪  skipping verifiers]
<CHOICE>: [your_choice]
"""

nightmare_following_question_prompt_with_reasoning_with_hint = """You chose
↪  Verifier <{verifier_num}> and the result is <{verifier_result}>.
```

```
**Hint**:
- `<PASS>` means your code satisfies the active criterion of the actual Verifier
↪  it is mapped to. For example, if Verifier 1 is secretly mapped to Verifier 2,
↪  then a <PASS> from Verifier 1 means your code met Verifier 2's hidden active
↪  rule.
- `<FAIL>` means your code does not satisfy the active criterion of the actual
↪  Verifier it is mapped to.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- Explain your reasoning step by step based on verifier result after <REASONING>
↪  tag, then provide your choice.

**Response format**:
<REASONING>: [Explain your reasoning step by step for choosing the verifier or
↪  skipping verifiers]
<CHOICE>: [your_choice]
"""

nightmare_after_last_question_prompt_with_reasoning_with_hint = """You chose
↪  Verifier <{verifier_num}> and the result is <{verifier_result}>.

You have now tested the maximum number of three verifiers for this round. The
↪  next stage is the Deduce Stage. If you want to test more verifiers or new
↪  code, you can choose SKIP during the Deduce Stage to move on to the next
↪  round.
"""

nightmare_not_valid_question_format_prompt_with_reasoning_with_hint = """You did
↪  not follow the required response format. Please try again with same choice.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_not_valid_verifier_choice_prompt_with_reasoning_with_hint = """You
↪  selected Verifier <{verifier_num}>, which is not a valid verifier number.

Please choose a valid verifier or SKIP to next stage.

**What You Must Do Now**:
- If you want to choose the next verifier to test, reply with verifier_num after
↪  <CHOICE> tag, such as <CHOICE>: 1.
- If you want to skip verifier testing for this round, reply with SKIP after
↪  <CHOICE> tag, such as <CHOICE>: SKIP.
- DO NOT include any explanation, only follow the response format.

**Response format**:
<CHOICE>: [your_choice]
"""

nightmare_deduce_prompt_with_reasoning_with_hint = """You are now entering the
↪  **Deduce Stage** of this round.

**Stage Purpose**:
In this stage, you can analyze all the information gathered then decide whether
↪  to submit a final guess or continue to the next round.

**Hint**:
```

```
- Passing all tested verifiers does not mean the code is correct if not all
↪  verifiers were tested. To be correct, the code must satisfy the hidden
↪  criteria of all verifiers, not just the ones you tested.
- You may choose not to test some verifiers if you can clearly reason that your
↪  code meets their requirements. But you must ensure every verifier is either
↪  tested and passed, or clearly justified through reasoning. Testing and
↪  passing only part of the verifiers is not enough if others are ignored.
- This stage **is not for testing**, you don't have to submit an answer; you can
↪  proceed to the next round to continue gathering information.
- Accuracy takes priority over speed. If you submit, the game will end, and an
↪  incorrect guess will result in immediate failure.

**Your Goal in This Stage**:
- Analysis all information gathered.
- Decide whether to submit the final guess or continue to the next round.
- Submission is not mandatory, you must make this decision based on your own
↪  reasoning.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  choice. If you want to submit a final guess, you must provide the reasons for
↪  not proceeding to the next round.

**Response format**:
<REASONING>: [Analysis and explain your reasoning step by step for continue to
↪  next round or submit final guess]
<CHOICE>: [your_choice]
"""

nightmare_deduce_result_prompt_with_reasoning_with_hint = "The final guess is
↪  {submitted_code}. The answer is {answer}, the guess is {is_correct}."

nightmare_not_valid_deduce_format_prompt_with_reasoning_with_hint = """You did
↪  NOT follow the response format. Please try again.

**What You Must Do Now**:
- If you want to continue to the next round, reply with SKIP after <CHOICE> tag,
↪  such as <CHOICE>: SKIP
- If you want to submit a final guess to end the game, reply with BLUE=X,
↪  YELLOW=Y, PURPLE=Z after <CHOICE> tag, such as <CHOICE>: BLUE=1, YELLOW=1,
↪  PURPLE=1.
- Explain your reasoning step by step with <REASONING> tag, then provide your
↪  choice.

**Response format**:
<REASONING>: [Analysis and explain your reasoning step by step for submitting the
↪  final guess or continue to next round]
<CHOICE>: [your_choice]
"""
% \end{tcolorbox}
% \end{verbatim}
```