

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 INVESTIGATING ADVANCED REASONING OF LARGE LANGUAGE MODELS VIA BLACK-BOX INTERACTION

Anonymous authors

Paper under double-blind review

ABSTRACT

Existing tasks fall short in evaluating reasoning ability of Large Language Models (LLMs) in an interactive, unknown environment. This deficiency leads to the isolated assessment of deductive, inductive, and abductive reasoning, neglecting the integrated reasoning process that is indispensable for humans discovery of real world. We introduce a novel evaluation paradigm, *black-box interaction*, to tackle this challenge. A black-box is defined by a hidden function that maps a specific set of inputs to outputs. LLMs are required to unravel the hidden function behind the black-box by interacting with it in given exploration turns, and reasoning over observed input-output pairs. Leveraging this idea, we build the ORACLE benchmark which comprises 6 types of black-box task and 96 black-boxes. 19 modern LLMs are benchmarked. o3, a leading LLM from OpenAI, ranks first in 5 of the 6 tasks, achieving over 70% accuracy on most easy black-boxes. But it still struggles with some hard black-box tasks, where its average performance drops below 40%. Further analysis indicates a universal difficulty among LLMs: They lack the high-level planning capability to develop efficient and adaptive exploration strategies for hypothesis refinement.

1 INTRODUCTION

Reasoning constitutes a fundamental component of artificial general intelligence (AGI), allowing systems to solve complex problems, adapt to unknown environment, and make decisions with human-like cognitive flexibility. With techniques like long chain-of-thought and test-time scaling (Chen et al., 2025), large language models (LLMs) (OpenAI, 2025b; Anthropic, 2025; Guo et al., 2025) have demonstrated remarkable reasoning ability in some challenging benchmarks (Cobbe et al., 2021; Hendrycks et al., 2021). However, skepticism has persistently shadowed the claim that LLMs possess reasoning ability akin to that of humans.

Charles Peirce’s framework (Peirce, 1934) posits that human’s discovery of unknown environment is guided by a dynamic reasoning cycle encompassing deduction, induction, and abduction. As depicted in Figure 1, this cycle begins with forming a hypothesis from observations (abduction), proceeds to planning to derive new observations (deduction), and concludes with hypothesis refinement against new observations (induction). However, existing reasoning datasets and benchmarks fall short in placing LLMs in an interactive, unknown environment (Fodor, 2025). This shortcoming leads to evaluating reasoning in an isolated manner, rather than an integrated, holistic process (Suzgun et al., 2022; Mondorf & Plank, 2024). Some researches (Costarelli et al., 2024; Hu et al., 2024) employ games to simulate interactive, unknown environments. This approach presents two key limitations. First, the extensive training data of LLMs raises the possibility that they are already familiar with the game strategy, compromising the validity of testing reasoning in *unknown* environment. Second, it conflates the evaluation of reasoning with other abilities like spatial understanding and long-context understanding, preventing it from serving as a pure reasoning benchmark.

To address the aforementioned challenge, we introduce *black-box interaction*, a novel evaluation paradigm for investigating the integrated, human-like reasoning capability of LLMs, which we term advanced reasoning. This paradigm models an unknown environment by constructing a black-box based on specific hidden rules. LLMs are required to uncover the hidden rules behind a black-box via multiple turns of exploration. Specifically, black-box is defined as a hidden function $f : \mathcal{X} \rightarrow \mathcal{Y}$, mapping input $\mathcal{X} = \{x | P(x)\}$ that satisfies predicate P to output \mathcal{Y} . LLMs are instructed to interact

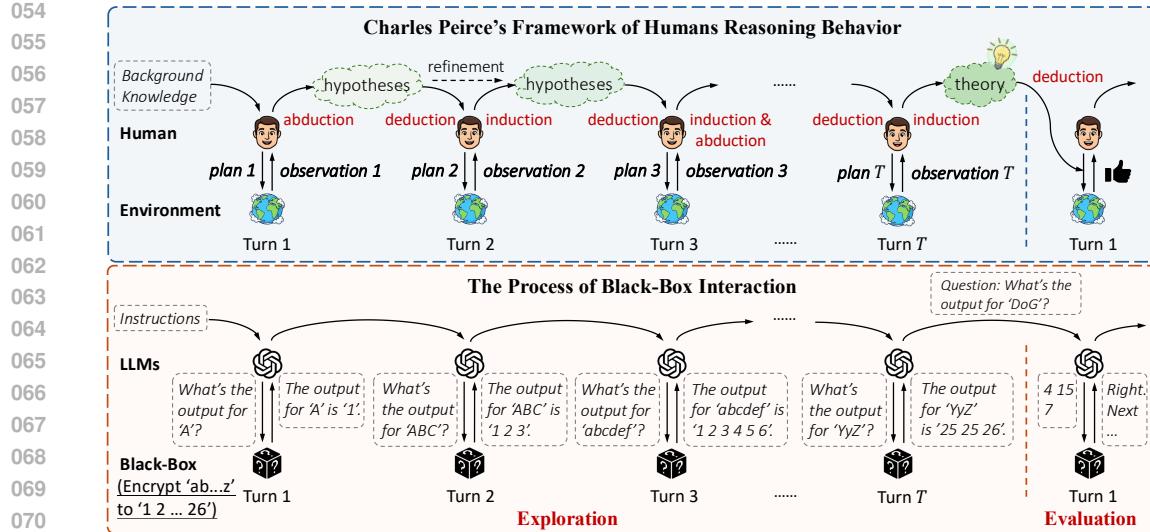


Figure 1: Illustration of Charles Peirce’s framework of human reasoning behavior (upper) and an example of the process of black-box interaction (lower). In this example, the black-box represents an encryption method that maps English letters to numbers.

with the black-box. The interaction is two-stage: (i) In the exploration stage, LLMs can freely feed any valid input x to the black-box, and will receive corresponding feedback $f(x)$. (ii) The evaluation stage starts after reaching given maximum exploration turns. LLMs’ understanding of the black-box is evaluated by comparing their output with the black-box’s output on a set of unseen test samples. Figure 1 illustrates the complete process of black-box interaction, where the black-box represents an encryption method that maps English letters to numbers.

The practical implementation of a black-box only involves mapping inputs to outputs based on hidden rules. This simplicity allows it to be generalized across various environments. To facilitate and accelerate the generalization of black-boxes to any scale, type, and level of difficulty, we design a fully automated agentic framework for black-box construction. Three LLM-based modules collaborate to accomplish black-box construction from scratch only with natural language description. It handles everything, including the generation of test samples, black-box code, and interactive interface between LLMs and black-box. Leveraging this framework, we build the ORACLE¹ benchmark, which considers 6 types of black-box task: **Code Intent Inference**, **Circuit Rule Inference**, **Physics System Inference**, **Encryption Rule Inference**, **Interactive Puzzle Inference**, **Game Strategy Inference**. The 6 tasks take code, boolean circuit, mechanical system, encryption method, interactive puzzle, opponent’s game strategy as black-box respectively. Current benchmark consists of 96 black-boxes, 51 of them are easy black-boxes and 45 are hard.

We evaluate 19 leading proprietary and open-weight LLMs. Overall, reasoning models perform better than chat models. OpenAI o3 delivers the best performance, ranking first in 5 out of 6 tasks under 10 exploration turns and 4 out of 6 tasks under 20 exploration turns. Furthermore, it achieves an average accuracy exceeding 70% on most easy black-boxes and approximately 40% on most hard ones. Further analysis reveals a critical and universal weakness of LLMs: They lack the high-level planning capability required to develop efficient and adaptive exploration strategies. This deficiency in reasoning prevents effective hypothesis refinement, which consequently compromises the ability to understand complex black-box mechanisms under limited exploration.

The contributions of this paper can be summarized as follows:

1. We introduce a novel evaluation paradigm, black-box interaction, for investigating advanced reasoning of LLMs (Section 2). This paradigm addresses several critical concerns in the field of reasoning dataset and benchmark design (Section 7).

¹The name is inspired by some mythologies that an ORACLE only returns “yes” or “no” to questions, thus challenging the questioner’s intelligence.

108 2. Leveraging the idea of black-box interaction, we build the ORACLE benchmark which
 109 comprises 6 types of black-box tasks and a total of 96 black-boxes (Section 3, Appendix G).
 110
 111 3. We propose an effective automated agentic framework which only requires natural lan-
 112 guage description to generate diverse black-boxes (Section 4). The framework greatly fa-
 113 cilitates the scaling of the ORACLE benchmark.
 114
 115 4. Comprehensive experiments and analysis are conducted to investigate the performance and
 116 behavior of LLMs in black-box interaction. We identify that LLMs struggle to develop
 117 efficient and adaptive exploration strategies. (Section 5, Appendix D, E).

118 **2 PRELIMINARIES**

120 We formally define the task setting of evaluating advanced reasoning of LLMs via black-box inter-
 121 action. A complete black-box interaction process comprises two sequential stages: an **exploration**
 122 stage and an **evaluation** stage.

124 **Black-Box** A black-box is a rule-based system characterized by a hidden function $f : \mathcal{X} \rightarrow \mathcal{Y}$
 125 that maps an input domain \mathcal{X} to an output domain \mathcal{Y} . The input domain \mathcal{X} is a set of elements
 126 $\mathcal{X} = \{x | P(x)\}$ that satisfy a specific predicate P . In some situations, f can be decomposed into a
 127 composition of multiple mappings, resulting in intermediate outputs. We define this more formally
 128 as follows: Let $\mathcal{Y}_0 = \mathcal{X}$. The composite function f is given by

$$f = f_n \circ f_{n-1} \circ \cdots \circ f_1, \quad (1)$$

131 where each component function $f_i : \mathcal{Y}_{i-1} \rightarrow \mathcal{Y}_i$ for $i = 1, \dots, n$. The intermediate outputs reside
 132 in the sets $\mathcal{Y}_1, \dots, \mathcal{Y}_{n-1}$, and the final output codomain is $\mathcal{Y} = \mathcal{Y}_n$.

134 **Model** The model, denoted by M , is a system that processes and generates natural language.
 135 Model M is instructed to interact with the black-box. Its action space is $\mathcal{X} = \{x | P(x)\}$. We focus
 136 on Transformer-based large language models in the scope of this paper.

138 **Exploration** The exploration stage consists of a sequence of interactions between model M and
 139 black-box f over T turns. In each turn $t \in \{1, \dots, T\}$, the model adaptively generates a query $x^t \in$
 140 \mathcal{X} and submits it to the black-box. It then receives the corresponding feedback $y^t = f(x^t) \in \mathcal{Y}$.
 141 In some scenarios, the model observes intermediate feedback $y_i^t \in \mathcal{Y}_i$ instead of y^t . The query x^t
 142 is generated based on the history of all previous interactions. Let the history at turn t be $H_{t-1} =$
 143 $(x^1, y^1, \dots, x^{t-1}, y^{t-1})$. The model generates the next query as:

$$x^t = M(H_{t-1}), \quad \text{for } t > 1. \quad (2)$$

145 The initial query, x^1 , is generated based on the initial task description provided to the model. Upon
 146 completion, this stage yields a total exploration history $H_T = (x^1, y^1, \dots, x^T, y^T)$.

148 **Evaluation** Following the exploration stage, the model’s reasoning ability is evaluated. This eval-
 149 uation stage runs for K turns, corresponding to the size of a test set $\mathcal{X}_{\text{test}} = \{x_{\text{test}}^1, \dots, x_{\text{test}}^K\}$. The
 150 test set is disjoint from the set of queries used during exploration, i.e., $\mathcal{X}_{\text{test}} \cap \{x^1, \dots, x^T\} = \emptyset$. In
 151 each turn $k \in \{1, \dots, K\}$, the model M is given a test sample x_{test}^k and needs to produce a predic-
 152 tion, denoted as \hat{y}^k . The black-box then provides feedback by comparing the prediction to the true
 153 output $f(x_{\text{test}}^k)$. This feedback is a binary correctness signal, $c^k = \mathbf{1}(\hat{y}^k = f(x_{\text{test}}^k))$, where $\mathbf{1}(\cdot)$ is
 154 the indicator function. The prediction at turn k is generated as:

$$\hat{y}^k = M(H_T, x_{\text{test}}^1, \hat{y}^1, c^1, \dots, x_{\text{test}}^{k-1}, \hat{y}^{k-1}, c^{k-1}, x_{\text{test}}^k), \quad \text{for } 1 \leq k \leq K. \quad (3)$$

157 This evaluation setup allows the model to continue to learn and adapt its strategy based on feedback
 158 received on its test-time performance.

161 **3 THE ORACLE BENCHMARK**

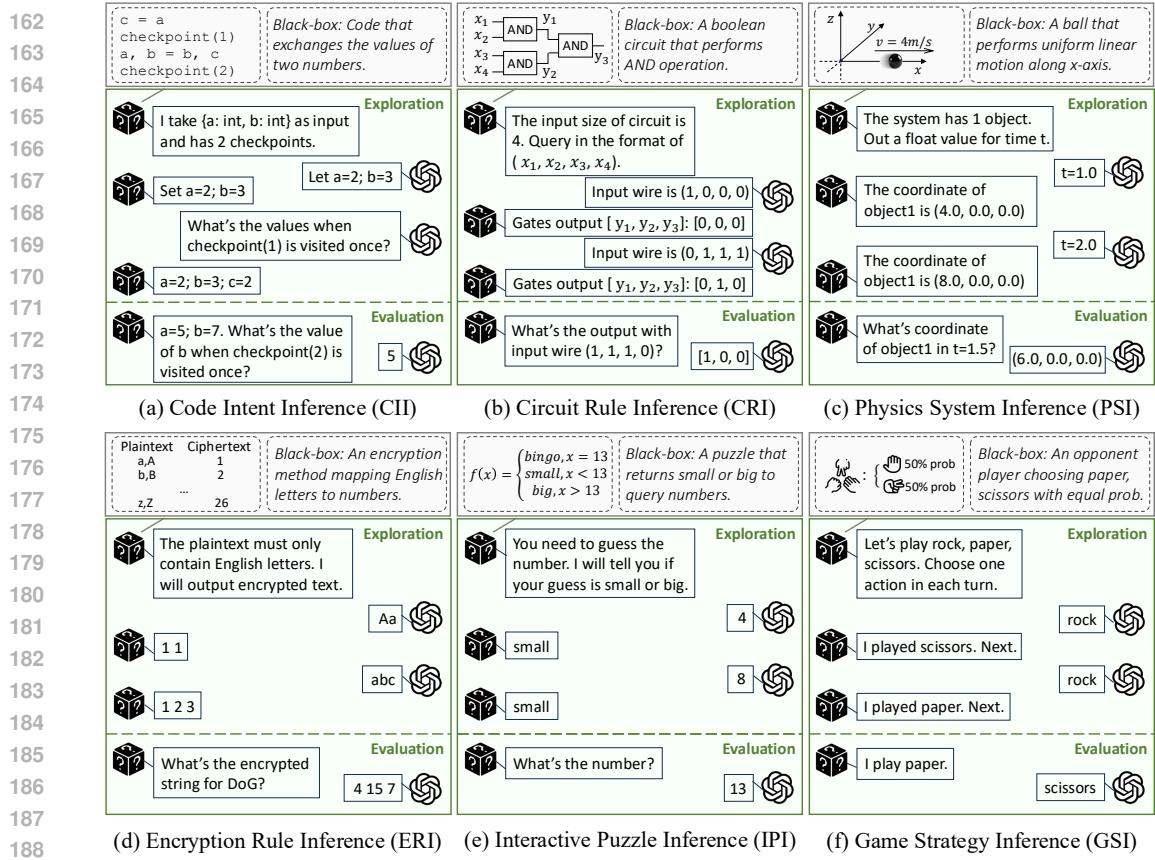


Figure 2: Examples of 6 different types of black-box tasks in the ORACLE benchmark.

The composition of ORACLE benchmark is shown in Figure 3. Each task consists of a mix of easy and hard black-boxes. The inner ring of the pie chart indicates the total number of black-boxes for each task, while the outer ring breaks down these black-boxes into easy and hard categories. The current benchmark includes 6 tasks and 96 black-boxes (51 easy, 45 hard).

3.1 TASK DESIGN

We reveal the methodology behind the design of 6 different black-box tasks. In Figure 2, a simple black-box from each task is selected to facilitate understanding. The examples cover both exploration and evaluation stages. Detailed implementations of all black-boxes in the ORACLE benchmark are reported in Appendix G. Some complete black-box interaction cases are shown in Appendix D.2. Test samples for each task are detailed in Appendix A.2.

Code Intent Inference (CII) A black-box f represents a code algorithm that maps input variables x to output variables y . Following the definition in Equation (1), f is further decomposed into f_i which is named checkpoint in this task. A checkpoint f_i captures the values of all current accessible variables. These checkpoints are strategically placed where significant changes to variable values occur. For LLMs, two types of actions are allowed: (i) Assign any valid value x as input variable. (ii) Ask for the value of accessible variables at selected checkpoint f_i . Action (i) must be completed before action (ii), and action (ii) is formatted in $(i, iter)$, where i is the index of selected checkpoint, and $iter$ is the visited times of the i -th checkpoint (e.g., within a loop). For example, $(3, 2)$ indicates the third checkpoint being visited for the second time. The goal of LLMs is to understand the algorithm. When evaluation starts, LLMs are required to output the value of questioned variable at certain checkpoint with unseen input variables.

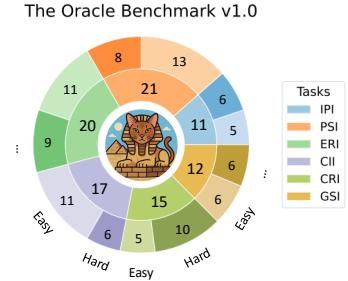


Figure 3: The composition of ORACLE benchmark.

Circuit Rule Inference (CRI) A black-box f represents an acyclic boolean circuit that only contains AND, OR, NOT gates. It maps input wire x which is a fixed number 0/1 bits to circuit gates' output y . The black-box will first inform the size n of input wire. Then in each turn, LLMs are supposed to output $x = (x_1, x_2, \dots, x_n)$, $x_i \in \{0, 1\}$ as query. After LLMs' query, the black-box will return the output of every circuit gate in the format of $y = [y_1, y_2, \dots, y_m]$, $y_i \in \{0, 1\}$, where m is the number of gates and y_i is the output of the i -th gate. The goal of LLMs is to understand the function and composition of circuit. When evaluation starts, LLMs are required to give every circuit's output with unseen input wires.

Physics System Inference (PSI) A black-box f represents a classical mechanical system that maps time point x to objects' coordinates y . In each turn, LLMs need to assign value x for time point as input, and the black-box will return the 3-dimensional coordinates y of all objects in the mechanical system at time x . The goal of LLMs is to understand the mechanical system. When evaluation starts, LLMs are required to calculate all the objects' coordinates at unseen time points.

Encryption Rule Inference (ERI) A black-box f represents an encryption process that maps plaintext x to ciphertext y . LLMs can assign any valid plaintext x as input, and black-box will return corresponding ciphertext y as output. The goal of LLMs is to understand how the encryption method works based on the plaintext-ciphertext pairs. When evaluation starts, LLMs are required to output the corresponding ciphertext given unseen plaintext.

Interactive Puzzle Inference (IPI) A black-box f represents an interactive puzzle with a hidden answer. The puzzle maps player query x to result y based on the puzzle rule. The LLMs can interact with the puzzle for multiple turns in the exploration stage. When evaluation starts, LLMs are required to figure out the right hidden answer of the puzzle.

Game Strategy Inference (GSI) Unlike the IPI task, the GSI task involves a two-player game. In this setup, LLMs participate as one player, facing a black-box opponent that performs a fixed game strategy. In this sense, the black-box f represents a strategy that maps game observation x to action y . Unlike previously introduced tasks, the GSI task requires a model to go beyond simply understanding the black-box: a model must devise a strategy to outperform it. When evaluation starts, LLMs will face the same black-box opponent and aim to achieve as higher score as possible. Since some games are not round-independent, one exploration turn in GSI indicates playing a n -round game once. Then LLMs will be evaluated in the game with the same number of rounds.

3.2 EVALUATION METRICS

Two metrics, accuracy and turn@shot, are used to measure the reasoning ability of LLMs in black-box interaction. Following the definitions in Section 2, the accuracy for each black-box is calculated via $acc = \sum_{k=1}^K c^k / K$, where K is the number of test samples and c^k measures the correctness of LLMs' answer. Specifically, accuracy in GSI task is measured by the ratio of actual score to the optimal strategy score. Turn@shot consists of two aspects. Turn denotes to the number of interaction turns for exploration, and shot indicates the number of allowed attempts for each test sample during evaluation. For example, 20@2 means the exploration stage lasts for 20 turns, and a model has 2 chances to answer each test sample in evaluation. The best model is supposed to achieve the highest accuracy with the lowest turn@shot.

4 FRAMEWORK FOR AUTOMATIC BLACK-BOX GENERATION

We introduce the agentic framework to generate diverse black-boxes for the ORACLE benchmark. As illustrated in Figure 4, the framework comprises three LLM-powered modules: a **Coding LLM** for initial creation of platform code, a **Test LLM** for interaction simulation, and a **Refinement LLM** for iterative debugging. The framework operates through the following three stages.

Platform Code Generation Platform code refers to the complete code for conducting black-box interaction, covering the implementation of black-box and interactive interface between LLMs and black-box. Leveraging the powerful coding capabilities of LLMs, we directly instruct a **Coding**

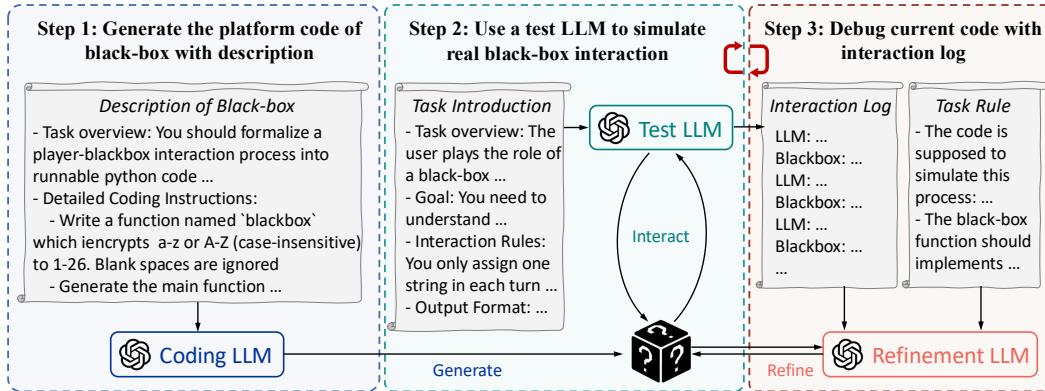


Figure 4: The framework for black-box generation, which is used to build the ORACLE benchmark. All related prompts are detailed in Appendix F.1.

LLM with prompt to generate the platform code. The prompt is twofold, encompassing natural language description of the black-box and the interaction rule. Since the interaction rule remains constant for certain type of black-box task, scaling up the benchmark simply involves describing the new black-boxes in natural language.

Simulation When initial platform code is generated, a **Test LLM** is used to interact with the black-box to simulate real interaction scenarios. This simulation covers both exploration and evaluation stage, and will result in three situations: (i) The platform code contains errors and fails to be executed. (ii) The platform code executes, but the black-box functionality is not correctly implemented as described. (iii) The platform code is correct and the simulation runs successfully as expected. In either case, an interaction log will be produced when the simulation process ends.

Iterative Debugging A **Refinement LLM** is used to check the correctness of generated platform code by combining the interaction log and task rule. The simulation process will result in three situations as mentioned above. For situation (i), Refinement LLM is instructed to produce revised platform code based on current code and error messages. For situation (ii), Refinement LLM is first instructed to figure out the inconsistency between current black-box implementation and its expected functionality with interaction log and task rule as prompt. Then, it's instructed to revise current platform code based on the discovered inconsistency. The simulation step will be conducted again when the platform code is revised. This iterative debugging process continues until the platform code is deemed correct by the Refinement LLM (situation (iii)).

The framework design operationalizes two principles from human cognition and software engineering: (i) Mastery through interaction, akin to learning a game by playing rather than just reading instructions. (ii) Debugging via runtime feedback, where code is refined based on its observed behavior rather than static analysis. This interactive, closed-loop process is key to generating high-fidelity code, and drastically facilitates the construction and scaling of the ORACLE benchmark.

5 EXPERIMENT AND ANALYSIS

5.1 BENCHMARKED MODELS AND BASELINE TEST

We benchmark a series of proprietary and open-weight models. Proprietary LLMs include GPT-series models (gpt-4o-mini, gpt-4o, gpt-4.1-mini, gpt-4.1, o1, o3-mini, o3, o4-mini), Claude-series models (claude-3.5-haiku, claude-3.5-sonnet, claude-3.7-sonnet, claude-4-sonnet), Gemini-series models (gemini-1.5-pro, gemini-2.0-flash, gemini-2.5-flash, gemini-2.5-pro), Qwen-series models (qwen-plus, qwen-max, qwq-plus). Open-weight LLMs include DeepSeek-series models (deepseek-v3-671b, deepseek-r1-671b), Llama-series models (llama-4-scout-17b-16e, llama-4-maverick-17b-128e), Qwen-series models (qwq-32b, qwen3-32b, qwen3-235b-a22b). See Appendix A for a complete list of models and implementation details. Some LLMs can perform extended thinking (i.e., reasoning). Both those with and without this capability are tested.

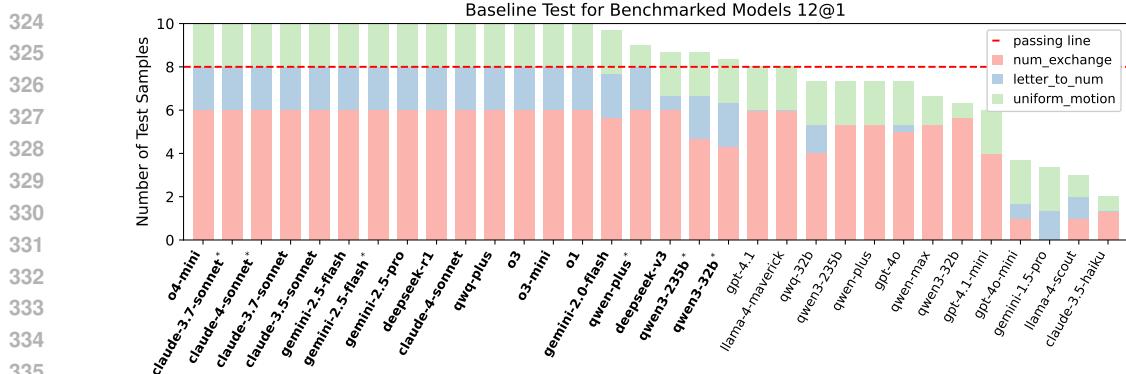


Figure 5: Baseline test for benchmarked models under 12@1. Models superscripted with * indicate extended thinking enabled. Qualified models are marked in bold.

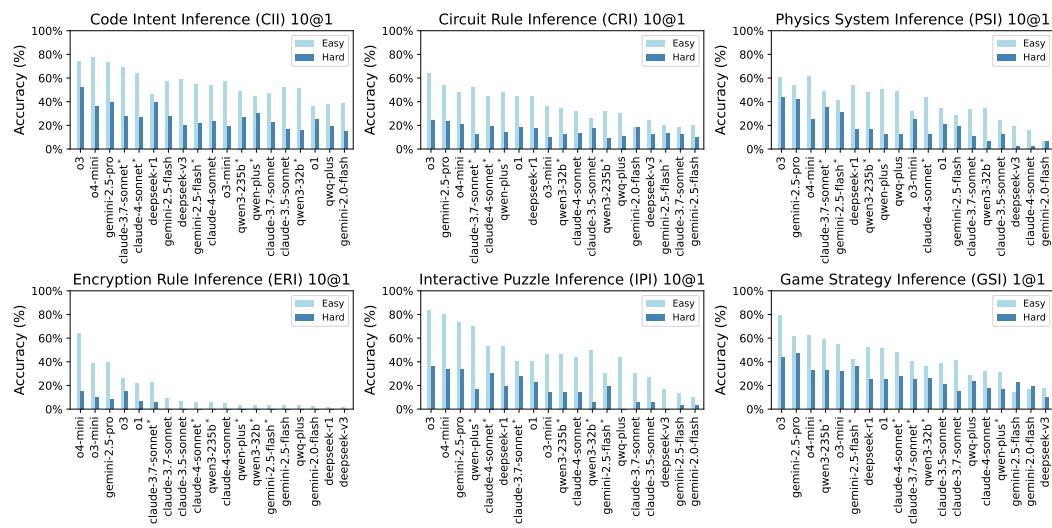


Figure 6: Performance of LLMs in six tasks of the ORACLE benchmark 10@1&1@1.

To qualify for the ORACLE benchmark, models must first pass a baseline test that contains 3 black-boxes from CII, ERI, PSI task, containing a total of 10 test samples. Detailed implementations of the 3 black-boxes are shown in Figure 2 (a), (c), (d). Turn@shot is set as 12@1, indicating 12 interaction turns for exploration and 1 chance for answering each test sample. The baseline test is conducted for three separate times and the averaged performance is reported in Figure 5. Models must achieve over 80% accuracy to qualify for the ORACLE benchmark. 19 out of 32 benchmarked models are qualified, including o1, o3-mini, o3, o4-mini, claude-3.5-sonnet, claude-3.7-sonnet, claude-3.7-sonnet_thinking, claude-4-sonnet, claude-4-sonnet_thinking, gemini-2.0-flash, gemini-2.5-flash, gemini-2.5-flash_thinking, gemini-2.5-pro, deepseek-v3, deepseek-r1, qwen-plus_thinking, qwq-plus, qwen3-235b-a22b_thinking, qwen3-32b_thinking.

5.2 OVERALL BENCHMARK RESULT

Two experiment settings, 10@1 and 20@2, are applied for the ORACLE benchmark. Figure 6 and Figure 7 report results on easy and hard black-boxes per task. Models are ranked by the sum of their accuracy on easy and hard black-boxes. Generally speaking, models exhibit similar rankings across 6 tasks. o3, o4-mini, gemini-2.5-pro, claude-3.7-sonnet_thinking, and claude-4-sonnet_thinking achieve competitive performance on all six tasks, and o3 ranks first among all models. When it comes to open-source models, deepseek-r1 achieves the top overall performance. Latest models (e.g., gemini-2.5-flash) perform better than old models (e.g., gemini-2.0-flash). Reasoning models (e.g., claude-4-sonnet_thinking) perform better than conventional chat models (e.g., claude-4-sonnet). While best performing LLMs boast over 80% accuracy on some easy black-box tasks, they

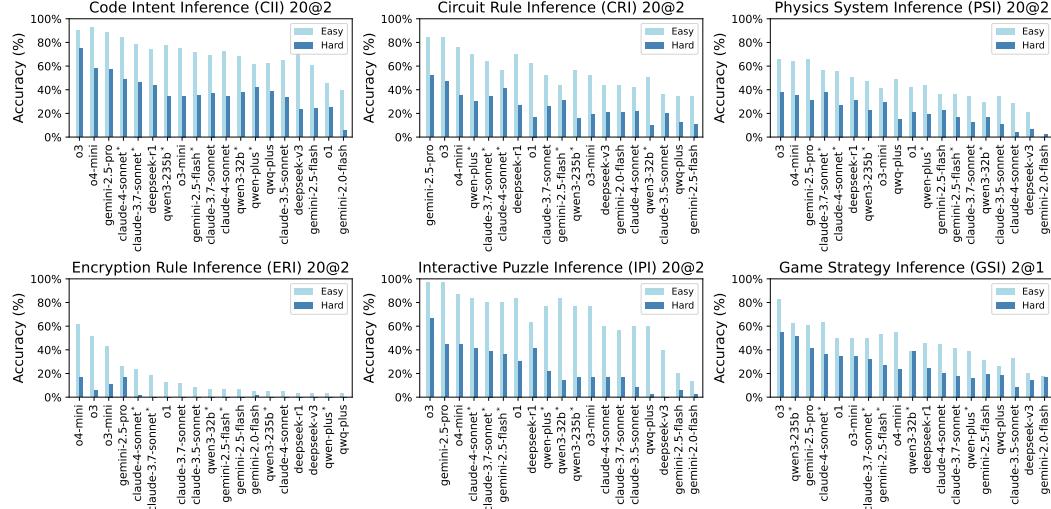


Figure 7: Performance of LLMs in six tasks of the ORACLE benchmark 20@2&2@1.

still struggle with harder ones, where their accuracy is typically less than half that of their performance on easy tasks.

5.3 ANALYSIS

We aim to reveal a key weakness of modern LLMs in black-box interaction: **They struggle to develop efficient and adaptive exploration strategies.** This deficiency in high-level planning highlights LLMs’ shortcomings in deductive and inductive reasoning. To substantiate this claim, we analyze the performance gains from increased exploration turns, present a comparative experiment, and examine the exploration behaviors of leading LLMs. Additional case analysis, ablation study, and weaknesses of LLMs are detailed in Appendix D, E, and C.4 respectively.

Analysis on performance gains from more exploration Model performance is expected to increase when exploration turns (from 10 to 20) and evaluation attempts (from 1 to 2) are extended. However, as shown in Figure 8, the averaged performance of LLMs improves by over 10% in CII, CRI, and IPI tasks, but it shows negligible improvement in PSI, ERI, and GSI tasks. While the limited progress on the PSI task is mainly due to the poor computing ability of LLMs (detailed in Appendix C.4), the lack of improvement on the ERI and GSI tasks highlights a fundamental weakness of LLMs: They are not good at developing efficient exploration strategy in some scenarios. We also find the performance gains is greater for easy black-boxes compared to hard ones, and this phenomenon becomes especially obvious when it comes to less capable LLMs. An example in Figure 8 indicates that the accuracy increase of deepseek-v3 remains near zero in hard black-boxes, while claudie-4-sonnet_thinking can still obtain great improvement. This suggests advanced models can devise and execute superior exploration strategies compared to less capable models.

Comparative experiment on adaptive exploration strategy optimization Apart from developing an efficient strategy, LLMs are supposed to keep optimizing a strategy adaptively based on instant feedback from black-box to narrow action space and maximize information gained from each turn, which is called adaptive exploration (Patrascu & Stacey, 1999). However, we find that even SOTA LLMs still lack a high-level planning ability to optimize exploration strategies. A comparative experiment with two settings is designed for verification. Setting (i): mod-

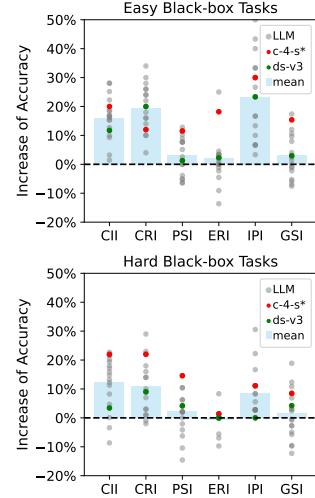


Figure 8: Averaged accuracy increase of 19 LLMs across 6 tasks when turn@shot is extended from 10@1&1@1 to 20@2&2@1.

	Setting (i) CRI 10@1	Setting (ii) CRI 10@1	Setting (i) ERI 10@1	Setting (ii) ERI 10@1
432	o4-mini: (0,0,0,0,0,0,0)	o4-mini: (0,0,0,0,0,0,0)	gemini-2.5-pro: A	gemini-2.5-pro: a
433	o4-mini: (0,0,0,0,0,0,0)	o4-mini: (1,0,0,0,0,0,0)	gemini-2.5-pro: a	gemini-2.5-pro: b
434	o4-mini: (1,0,0,0,0,0,0)	o4-mini: (0,1,0,0,0,0,0)	gemini-2.5-pro: B	gemini-2.5-pro: c
435	o4-mini: (0,1,0,0,0,0,0)	o4-mini: (1,1,0,0,0,0,0)	gemini-2.5-pro: b	gemini-2.5-pro: z
436	o4-mini: (0,0,1,0,0,0,0)	o4-mini: (0,0,1,0,0,0,0)	gemini-2.5-pro: C	gemini-2.5-pro: Hello
437	o4-mini: (0,0,0,1,0,0,0)	o4-mini: (0,0,0,1,0,0,0)	gemini-2.5-pro: c	gemini-2.5-pro: word
438	o4-mini: (0,0,0,0,1,0,1)	o4-mini: (0,0,0,0,1,0,0)	gemini-2.5-pro: Z	gemini-2.5-pro: book
439	o4-mini: (1,1,1,1,1,1,1)	o4-mini: (0,0,0,0,1,0,0)	gemini-2.5-pro: z	gemini-2.5-pro: cat
440	o4-mini: (0,0,0,1,0,0,0)	o4-mini: (0,0,0,0,0,1,0)	gemini-2.5-pro: Hello	gemini-2.5-pro: in
441	o4-mini: (0,0,0,0,1,0,0)	o4-mini: (0,0,0,0,0,1,0)	gemini-2.5-pro: Apple Bee	gemini-2.5-pro: banana
442	<i>Final Accuracy: 0%</i>		<i>Final Accuracy: 0%</i>	

Figure 9: Cases of exploration behavior under two different settings. Black-box responses and evaluation stages are neglected. Red text indicates the same exploration behavior.

els will not receive black-box feedback in each turn. Instead, all the queries and corresponding answers will be announced in the last exploration turn; Setting (ii) serves as a control group, where models will receive instant black-box feedback in each turn. Ideally, model performance under setting (ii) is supposed to be higher than setting (i), as models can keep optimizing exploration strategy with instant feedback, but they have to maintain a fixed strategy in setting (i). We select three powerful LLMs, gemini-2.5-pro, o3-mini, and o4-mini, and evaluate them in two representative tasks, CRI and ERI, which most challenge models’ ability in optimizing exploration strategy. Results are shown in Figure 10. The three LLMs exhibit remarkably consistent performance across the two settings, providing strong evidence of their inability to optimize exploration strategies effectively. To further investigate the exploration strategy of LLMs under two different settings, we show some cases of LLMs’ exploration behavior in Figure 9. These cases come from LLMs’ interaction with two easy black-boxes from CRI and ERI task (“Xor Sequence” and “Zigzag Cipher”, detailed in Appendix G). First, we find both models adopt inefficient exploration strategies: In the CRI task, o4-mini employs an exhaustive, in-order strategy. In the ERI task, gemini-2.5-pro resorts to querying single English letter or word. Second, both models fail to adaptively optimize exploration strategy. Their reasoning behavior remains largely consistent across two settings, indicating that they cannot effectively exploit the real-time feedback from the black-box. Consequently, both models achieve zero accuracy in evaluation.

Building on the analysis above, we categorize the capacity to devise exploration strategy into three tiers. **Tier 1**: Model can not develop a planned exploration strategy, and explore in a random approach. **Tier 2**: Model can develop a relatively efficient exploration strategy but fail to optimize it adaptively. **Tier 3**: Model can adaptively optimize their exploration strategy based on instant feedback, developing a nearly optimal approach. Most LLMs operate at Tier 1. Best-performed reasoning LLMs achieve Tier 2 in some situations. Tier 3 is the domain of human according to Charles Peirce’s theory and primitive human evaluation (Appendix E.7). We have not yet identified any LLM that can achieve Tier 3 of adaptive strategy planning.

6 RELATED WORK

6.1 MODELING INTERACTIVE ENVIRONMENT

Building interactive environment that simulates real-world settings has always been a heated research topic. Prior works in reinforcement learning build online (Brockman et al., 2016) and offline (Fu et al., 2020) environment to investigate models’ ability of strategic learning. Recent progress

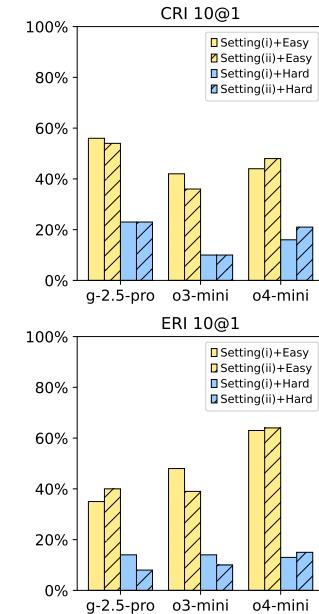


Figure 10: Model performance in CRI 10@1 and ERI 10@1 under two settings.

486 in evaluating LLMs and LLM agents has adopted various methods to model interactive environment.
 487 For example, WebArena (Zhou et al., 2023) creates an environment with fully functional
 488 websites that contain tools and external knowledge bases. Fish et al. (2025) builds stationary and
 489 non-stationary economic environment. Wu et al. (2023); Costarelli et al. (2024); Hu et al. (2024);
 490 Park et al. (2025) employ text games (e.g. Akinator) or video games (e.g. MineCraft) as environment
 491 and evaluate LLMs’ reasoning ability through game-playing. Ma et al. (2024) propose AgentBoard
 492 which contains web, tool, embodied AI, and game tasks as partially observable environments.

494 6.2 REASONING DATASETS AND BENCHMARKS

496 Evaluating reasoning ability of LLMs is an active area of research, especially with the recent develop-
 497 ment of reasoning large language models (Chen et al., 2025). In the field of deductive reasoning,
 498 several datasets and benchmarks are developed for measuring complex mathematics (e.g. GSM8k
 499 (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), AIME, AMC23, Omni-MATH (Gao et al.,
 500 2024), FrontierMath (Glazer et al., 2024), OlympiadBench (He et al., 2024)), coding (e.g. Code-
 501 Contests (Li et al., 2022), SWEbench (Jimenez et al., 2023), LiveCodeBench (Jain et al., 2024)), and
 502 logic (e.g. BIGBench Hard (Suzgun et al., 2022), LiveBench (White et al., 2024), ARC (Chollet,
 503 2019), ZebraLogic (Lin et al., 2025)). Datasets for inductive reasoning include DEER (Yang et al.,
 504 2022), ConceptARC (Moskvichev et al., 2023), Mirage (Li et al., 2024), InductionBench (Hua et al.,
 505 2025), Abductive reasoning datasets include ART (Bhagavatula et al., 2020), CauseLogics (He &
 506 Lu, 2024). Some researches like UniADILR (Xia et al., 2025) seek to evaluate deductive, inductive,
 507 and abductive reasoning in one framework.

509 7 DISCUSSION

511 The design of black-box interaction can bring additional benefits. First, it addresses the critical
 512 concern of data contamination (Roberts et al., 2023; Deng et al., 2023), which refers to the leakage
 513 of datasets and benchmarks into LLMs’ training data, thus hindering the discrimination of whether
 514 LLMs truly reason or just memorize (Magar & Schwartz, 2022; Zhang et al., 2022; Dziri et al., 2023;
 515 Wu et al., 2024; Balloccu et al., 2024). Black-box interaction naturally generate dynamic context as
 516 input. The inherent invisibility of black-box also ensures zero data contamination, even if LLMs are
 517 highly acquainted with its practical implementation. We also conduct an ablation study (Appendix
 518 E.3) which directly adds relevant knowledge as prompt for verification.

519 Second, it facilitates the evaluation process. Previous works (Turpin et al., 2023; Hao et al., 2024;
 520 Mondorf & Plank, 2024) find LLMs can generate correct answers with logically incorrect reason-
 521 ing paths. Thus evaluations on most outcome-based datasets and benchmarks (Cobbe et al., 2021;
 522 Hendrycks et al., 2021) become less convincing. In the scenario of black-box interaction, the inter-
 523 action history naturally reflects the reasoning path of LLMs, and an incorrect reasoning path will
 524 not lead to correctness in all test samples. So evaluation on test samples is a reliable approach.

525 Third, black-box interaction simulates many concrete real-world applications. For example, in the
 526 designed six tasks in the ORACLE benchmark, CII models code reverse engineering or decipher-
 527 ing legacy code via breakpoint debugging. CRI models hardware reverse engineering or integrated
 528 circuit (IC) analysis. PSI models scientific discovery and the scientific method. ERI models crypt-
 529 analysis under the Chosen-Plaintext Attack (CPA) setting. IPI models interactive problem-solving.
 530 GSI models opponent modeling and devising counter-strategies within game-theoretic scenarios.

532 8 CONCLUSION

535 We introduce black-box interaction, a novel paradigm for interactively evaluating the advanced rea-
 536 soning of LLMs, and propose the corresponding ORACLE benchmark. This benchmark features 6
 537 task designs and 96 black-boxes to evaluate 19 modern LLMs. The ORACLE benchmark is highly
 538 adaptable, allowing for easy scaling to any scale, task, and difficulty level through the a robust
 539 agentic generation framework. We also provide deep insight into the reasoning behavior and short-
 540 comings of current LLMs in uncovering the hidden rules behind the black-box.

540 REPRODUCIBILITY STATEMENT
541542 We strive to make sure all the results reported in this paper are reproducible. The source code has
543 been submitted as supplementary materials.
544545 THE USE OF LARGE LANGUAGE MODELS (LLMs)
546548 Large Language Models (LLMs) are only used for polish writing in this paper. Specifically, LLMs
549 are exclusively used to rewrite individual sentences based on original human text to improve gram-
550 mar and wording. The authors of this paper are responsible for the contents of their submission.
551552 REFERENCES
553

554 Anthropic. The claude 3 model family: Opus, sonnet, haiku. 2024. (Cited on pg. 18)

555 Anthropic. Claude 3.7 sonnet system card. 2025. (Cited on pg. 1, 18)

557 Simone Balloccu, Patrícia Schmidlová, Mateusz Lango, and Ondřej Dušek. Leak, cheat, re-
558 peat: Data contamination and evaluation malpractices in closed-source llms. *arXiv preprint*
559 *arXiv:2402.03927*, 2024. (Cited on pg. 10, 19)560 Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Han-
561 nah Rashkin, Doug Downey, Wen-tau Yih, and Yejin Choi. Abductive commonsense reasoning.
562 In *International Conference on Learning Representations*, 2020. (Cited on pg. 10)564 Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and
565 Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. (Cited on pg. 9)567 Arthur W Burks. Peirce's theory of abduction. *Philosophy of science*, 13(4):301–306, 1946. (Cited
568 on pg. 19)570 Boxi Cao, Mengjie Ren, Hongyu Lin, Xianpei Han, Feng Zhang, Junfeng Zhan, and Le Sun. Structe-
571 val: Deepen and broaden large language model assessment via structured evaluation. *arXiv*
572 *preprint arXiv:2408.03281*, 2024. (Cited on pg. 19)573 Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu,
574 Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-
575 thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025. (Cited on
576 pg. 1, 10)577 François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019. (Cited
578 on pg. 10)580 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,
581 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to
582 solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. (Cited on pg. 1, 10, 19)584 Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit
585 Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the
586 frontier with advanced reasoning, multimodality, long context, and next generation agentic capa-
587 bilities. *arXiv preprint arXiv:2507.06261*, 2025. (Cited on pg. 18)588 Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng,
589 Wenjie Li, Joshua Clymer, and Arjun Yadav. Gamebench: Evaluating strategic reasoning abilities
590 of llm agents. *arXiv preprint arXiv:2406.06613*, 2024. (Cited on pg. 1, 10)592 Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. Investigat-
593 ing data contamination in modern benchmarks for large language models. *arXiv preprint*
594 *arXiv:2311.09783*, 2023. (Cited on pg. 10, 19)

594 Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean
 595 Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of trans-
 596 formers on compositionality. *Advances in Neural Information Processing Systems*, 36:70293–
 597 70332, 2023. (Cited on pg. 10)

598 Kuang Tih Fann. *Peirce’s theory of abduction*. Springer Science & Business Media, 2012. (Cited
 599 on pg. 19)

600 Sara Fish, Julia Shephard, Minkai Li, Ran I Shorrer, and Yannai A Gonczarowski. Econevals:
 601 Benchmarks and litmus tests for llm agents in unknown environments. *arXiv preprint*
 602 *arXiv:2503.18825*, 2025. (Cited on pg. 10)

603 James Fodor. Line goes up? inherent limitations of benchmarks for evaluating large language
 604 models. *arXiv preprint arXiv:2502.14318*, 2025. (Cited on pg. 1)

605 Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep
 606 data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020. (Cited on pg. 9)

607 Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma,
 608 Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark
 609 for large language models. *arXiv preprint arXiv:2410.07985*, 2024. (Cited on pg. 10)

610 GeminiTeam, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer,
 611 Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal under-
 612 standing across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. (Cited on
 613 pg. 18)

614 Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caro-
 615 line Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Fron-
 616 tiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint*
 617 *arXiv:2411.04872*, 2024. (Cited on pg. 10)

618 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
 619 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
 620 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. (Cited on pg. 1, 18)

621 Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma,
 622 Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhitong Hu. Llm reasoners: New evaluation,
 623 library, and analysis of step-by-step reasoning with large language models, 2024. (Cited on pg.
 624 10, 19)

625 Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu,
 626 Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for
 627 promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint*
 628 *arXiv:2402.14008*, 2024. (Cited on pg. 10)

629 Jinwei He and Feng Lu. Causejudger: Identifying the cause with llms for abductive logical reason-
 630 ing. *arXiv preprint arXiv:2409.05559*, 2024. (Cited on pg. 10)

631 Kaiyu He, Mian Zhang, Shuo Yan, Peilin Wu, and Zhiyu Chen. Idea: Enhancing the rule learning
 632 ability of large language model agent through induction, deduction, and abduction. In *Findings*
 633 of the Association for Computational Linguistics: ACL 2025, pp. 13563–13597, 2025. (Cited on
 634 pg. 20)

635 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,
 636 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv*
 637 *preprint arXiv:2103.03874*, 2021. (Cited on pg. 1, 10, 19)

638 Lanxiang Hu, Qiyu Li, Anze Xie, Nan Jiang, Ion Stoica, Haojian Jin, and Hao Zhang. Gamearena:
 639 Evaluating llm reasoning through live computer games. *arXiv preprint arXiv:2412.06394*, 2024.
 640 (Cited on pg. 1, 10, 20)

648 Wenyue Hua, Tyler Wong, Sun Fei, Liangming Pan, Adam Jardine, and William Yang Wang. Induc-
 649 tionbench: Llms fail in the simplest complexity class. *arXiv preprint arXiv:2502.15823*, 2025.
 650 (Cited on pg. 10)

651

652 Kaixuan Huang, Jiacheng Guo, Zihao Li, Xiang Ji, Jiawei Ge, Wenzhe Li, Yingqing Guo, Tianle Cai,
 653 Hui Yuan, Runzhe Wang, Yue Wu, Ming Yin, Shange Tang, Yangsibo Huang, Chi Jin, Xinyun
 654 Chen, Chiyuan Zhang, and Mengdi Wang. Math-perturb: Benchmarking llms' math reasoning
 655 abilities against hard perturbations, 2025. (Cited on pg. 19)

656 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
 657 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint*
 658 *arXiv:2410.21276*, 2024. (Cited on pg. 18)

659

660 Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec
 661 Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv*
 662 *preprint arXiv:2412.16720*, 2024. (Cited on pg. 18)

663

664 Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando
 665 Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free
 666 evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. (Cited on
 667 pg. 10, 19)

668

669 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik
 670 Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint*
 671 *arXiv:2310.06770*, 2023. (Cited on pg. 10)

672

673 Eldar Kurtic, Amir Moeini, and Dan Alistarh. Mathador-lm: A dynamic benchmark for mathemat-
 674 ical reasoning on large language models. *arXiv preprint arXiv:2406.12572*, 2024. (Cited on pg.
 675 19)

676

677 Jinu Lee and Julia Hockenmaier. Evaluating step-by-step reasoning traces: A survey. *arXiv preprint*
 678 *arXiv:2502.12289*, 2025. (Cited on pg. 19)

679

680 Fangyu Lei, Qian Liu, Yiming Huang, Shizhu He, Jun Zhao, and Kang Liu. S3eval: A synthetic,
 681 scalable, systematic evaluation suite for large language models. *arXiv preprint arXiv:2310.15147*,
 682 2023. (Cited on pg. 19)

683

684 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal,
 685 Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented gener-
 686 ation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:
 687 9459–9474, 2020. (Cited on pg. 19)

688

689 Jiachun Li, Pengfei Cao, Zhuoran Jin, Yubo Chen, Kang Liu, and Jun Zhao. Mirage: Evaluating and
 690 explaining inductive reasoning process in language models. *arXiv preprint arXiv:2410.09542*,
 691 2024. (Cited on pg. 10)

692

693 Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom
 694 Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation
 695 with alphacode. *Science*, 378(6624):1092–1097, 2022. (Cited on pg. 10)

696

697 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan
 698 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth*
 699 *International Conference on Learning Representations*, 2023. (Cited on pg. 19)

700

701 Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter
 702 Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv*
 703 *preprint arXiv:2502.01100*, 2025. (Cited on pg. 10)

704

705 Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao,
 706 Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint*
 707 *arXiv:2412.19437*, 2024. (Cited on pg. 18)

702 Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Ling-
 703 peng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents.
 704 *arXiv preprint arXiv:2401.13178*, 2024. (Cited on pg. 10)

705 Inbal Magar and Roy Schwartz. Data contamination: From memorization to exploitation. *arXiv*
 706 *preprint arXiv:2203.08242*, 2022. (Cited on pg. 10, 19)

708 Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao.
 709 Leveraging online olympiad-level math problems for Ilms training and contamination-resistant
 710 evaluation. *arXiv preprint arXiv:2501.14275*, 2025. (Cited on pg. 19)

711 Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad
 712 Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large
 713 language models. *arXiv preprint arXiv:2410.05229*, 2024. (Cited on pg. 19)

715 Philipp Mondorf and Barbara Plank. Beyond accuracy: Evaluating the reasoning behavior of large
 716 language models - a survey. In *First Conference on Language Modeling*, 2024. (Cited on pg. 1,
 717 10, 19)

718 Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The conceptarc benchmark:
 719 Evaluating understanding and generalization in the arc domain. *arXiv preprint arXiv:2305.07141*,
 720 2023. (Cited on pg. 10)

722 OpenAI. Introducing gpt-4.1 in the api. 2025a. (Cited on pg. 18)

723 OpenAI. Openai o3 and o4-mini system card. 2025b. (Cited on pg. 1, 18)

725 Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu
 726 Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn, et al. Orak: A foundational benchmark
 727 for training and evaluating llm agents on diverse video games. *arXiv preprint arXiv:2506.03610*,
 728 2025. (Cited on pg. 10)

729 Relu Patrascu and Deborah Stacey. Adaptive exploration in reinforcement learning. In *IJCNN'99.*
 730 *International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, vol-
 731 ume 4, pp. 2276–2281. IEEE, 1999. (Cited on pg. 8)

733 Charles Sanders Peirce. *Collected papers of charles sanders peirce*, volume 5. Harvard University
 734 Press, 1934. (Cited on pg. 1, 19)

735 Kun Qian, Shunji Wan, Claudia Tang, Youzhi Wang, Xuanming Zhang, Maximillian Chen, and Zhou
 736 Yu. Varbench: Robust language model benchmarking through dynamic variable perturbation.
 737 *arXiv preprint arXiv:2406.17681*, 2024. (Cited on pg. 19)

739 QwenTeam. Qwq-32b: Embracing the power of reinforcement learning. 2025. (Cited on pg. 18)

740 Manley Roberts, Himanshu Thakur, Christine Herlihy, Colin White, and Samuel Dooley. To the
 741 cutoff... and beyond? a longitudinal perspective on llm data contamination. In *The Twelfth Inter-*

742 *national Conference on Learning Representations*, 2023. (Cited on pg. 10, 19)

744 Jiajun Shi, Jian Yang, Jiaheng Liu, Xingyuan Bu, Jiangjie Chen, Junting Zhou, Kaijing Ma, Zhou-
 745 futu Wen, Bingli Wang, Yancheng He, et al. Korgym: A dynamic game platform for llm reasoning
 746 evaluation. *arXiv preprint arXiv:2505.14552*, 2025. (Cited on pg. 20)

747 Saurabh Srivastava, Anto PV, Shashank Menon, Ajay Sukumar, Alan Philipose, Stevin Prince,
 748 Sooraj Thomas, et al. Functional benchmarks for robust evaluation of reasoning performance,
 749 and the reasoning gap. *arXiv preprint arXiv:2402.19450*, 2024. (Cited on pg. 19)

751 Shichao Sun, Junlong Li, Weizhe Yuan, Ruifeng Yuan, Wenjie Li, and Pengfei Liu. The critique of
 752 critique. *arXiv preprint arXiv:2401.04518*, 2024. (Cited on pg. 19)

753 Mirac Suzgun, Nathan Scales, Nathanael Schärdli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung,
 754 Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks
 755 and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022. (Cited on
 pg. 1, 10)

756 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée
 757 Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and
 758 efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. (Cited on pg. 18)
 759

760 Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always
 761 say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural
 762 Information Processing Systems*, 36:74952–74965, 2023. (Cited on pg. 10, 19)

763 Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun.
 764 Towards understanding chain-of-thought prompting: An empirical study of what matters. *arXiv
 765 preprint arXiv:2212.10001*, 2022. (Cited on pg. 19)

766 Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-
 767 Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-
 768 free llm benchmark. *arXiv preprint arXiv:2406.19314*, 4, 2024. (Cited on pg. 10, 19)
 769

770 Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. Smartplay: A benchmark for llms as
 771 intelligent agents. *arXiv preprint arXiv:2310.01557*, 2023. (Cited on pg. 10, 20)

772 Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim,
 773 Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations
 774 of language models through counterfactual tasks. In *Proceedings of the 2024 Conference of the
 775 North American Chapter of the Association for Computational Linguistics: Human Language
 776 Technologies (Volume 1: Long Papers)*, pp. 1819–1862, 2024. (Cited on pg. 10)
 777

778 Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. Evaluating mathematical
 779 reasoning beyond accuracy. In *Proceedings of the AAAI Conference on Artificial Intelligence*,
 780 volume 39, pp. 27723–27730, 2025. (Cited on pg. 10)

781 An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li,
 782 Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint
 783 arXiv:2412.15115*, 2024. (Cited on pg. 18)

784 Zonglin Yang, Li Dong, Xinya Du, Hao Cheng, Erik Cambria, Xiaodong Liu, Jianfeng Gao, and
 785 Furu Wei. Language models as inductive reasoners. *arXiv preprint arXiv:2212.10923*, 2022.
 786 (Cited on pg. 10)
 787

788 Jiahao Ying, Yixin Cao, Yushi Bai, Qianru Sun, Bo Wang, Wei Tang, Zhaojun Ding, Yizhe Yang,
 789 Xuanjing Huang, and Shuicheng Yan. Automating dataset updates towards reliable and timely
 790 evaluation of large language models. *arXiv preprint arXiv:2402.11894*, 2024. (Cited on pg. 19)

791 Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck. On the
 792 paradox of learning to reason from data. *arXiv preprint arXiv:2205.11502*, 2022. (Cited on pg.
 793 10)
 794

795 Zhehao Zhang, Jiaao Chen, and Diyi Yang. Darg: Dynamic evaluation of large language models via
 796 adaptive reasoning graph. *arXiv preprint arXiv:2406.17271*, 2024. (Cited on pg. 19)

797 Qihao Zhao, Yangyu Huang, Tengchao Lv, Lei Cui, Qinzheng Sun, Shaoguang Mao, Xin Zhang,
 798 Ying Xin, Qifeng Yin, Scarlett Li, et al. Mmlu-cf: A contamination-free multi-task language
 799 understanding benchmark. *arXiv preprint arXiv:2412.15194*, 2024. (Cited on pg. 19)
 800

801 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,
 802 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chat-
 803 bot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023. (Cited
 804 on pg. 19)

805 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
 806 Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for build-
 807 ing autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. (Cited on pg. 10)

808 Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. Dyval:
 809 Dynamic evaluation of large language models for reasoning tasks, 2024. (Cited on pg. 19)

810	APPENDIX	
811		
812		
813	A Implementation Details	18
814	A.1 Details of LLMs	18
815	A.2 Details of ORACLE benchmark	18
816		
817		
818	B Additional Related Work	19
819	B.1 Data Contamination and Dynamic Benchmark	19
820	B.2 Evaluation of Reasoning Ability	19
821	B.3 Charles Peirce’s Framework of Humans Reasoning Behavior	19
822		
823		
824	C Additional Discussion	20
825	C.1 Limitations	20
826	C.2 Comparison with Previous Work	20
827	C.3 More Task Settings for Black-Box Interaction	20
828		
829	C.4 Extra Findings	21
830		
831		
832	D Case Study	22
833	D.1 How Iterative Debugging Works	22
834	D.2 How LLMs Interact with the Black-Box	27
835		
836	D.3 How LLMs Succeed and Fail	31
837		
838		
839	E Ablation Study	42
840	E.1 The Influence of Temperature	42
841	E.2 The Influence of Extended Thinking	43
842		
843	E.3 The Influence of Prior Knowledge	43
844		
845	E.4 The Influence of Exploration Turns	44
846		
847	E.5 The Influence of Reasoning Tokens	44
848		
849	E.6 The Influence of Reflective Instruction	44
850		
851	E.7 Human Evaluation	44
852		
853	F Prompt Details	45
854	F.1 Prompt for Black-Box Generation	45
855	F.2 Prompt for Black-Box Interaction	51
856		
857	G Details of Black-Boxes in the ORACLE benchmark	53
858	G.1 Code Intent Inference (CII)	54
859	G.2 Circuit Rule Inference (CRI)	55
860		
861	G.3 Physics System Inference (PSI)	56
862		
863	G.4 Encryption Rule Inference (ERI)	59
864		
865	G.5 Interactive Puzzle Inference (IPI)	62

864	G.6 Game Strategy Inference (GSI)	64
865		
866		
867	H Detailed Experimental Results	67
868		
869		
870		
871		
872		
873		
874		
875		
876		
877		
878		
879		
880		
881		
882		
883		
884		
885		
886		
887		
888		
889		
890		
891		
892		
893		
894		
895		
896		
897		
898		
899		
900		
901		
902		
903		
904		
905		
906		
907		
908		
909		
910		
911		
912		
913		
914		
915		
916		
917		

918
919
Table 1: Model code/API of benchmarked models.
920

Model Name	Model Type	API Access
GPT-4o-mini (Hurst et al., 2024)	Proprietary	gpt-4o-mini-2024-07-18
GPT-4o (Hurst et al., 2024)	Proprietary	gpt-4o-2024-08-06
GPT-4.1-mini (OpenAI, 2025a)	Proprietary	gpt-4.1-mini-2025-04-14
GPT-4.1 (OpenAI, 2025a)	Proprietary	gpt-4.1-2025-04-14
o1 (Jaech et al., 2024)	Proprietary	o1-2024-12-17
o3-mini (OpenAI, 2025b)	Proprietary	o3-mini-2025-01-31
o3 (OpenAI, 2025b)	Proprietary	o3-2025-04-16
o4-mini (OpenAI, 2025b)	Proprietary	o4-mini-2025-04-16
Claude-3.5-haiku (Anthropic, 2024)	Proprietary	claude-3-5-haiku-20241022
Claude-3.5-sonnet (Anthropic, 2024)	Proprietary	claude-3-5-sonnet-20241022
Claude-3.7-sonnet (Anthropic, 2025)	Proprietary	claude-3-7-sonnet-20250219
Claude-4-sonnet (Anthropic, 2025)	Proprietary	claude-sonnet-4-20250514
Gemini-1.5-pro (GeminiTeam et al., 2024)	Proprietary	gemini-1.5-pro
Gemini-2.0-flash (GeminiTeam et al., 2024)	Proprietary	gemini-2.0-flash
Gemini-2.5-flash (Comanici et al., 2025)	Proprietary	gemini-2.5-flash
Gemini-2.5-pro (Comanici et al., 2025)	Proprietary	gemini-2.5-pro
DeepSeek-v3-671b (Liu et al., 2024)	Open-weight	deepseek-reasoner
DeepSeek-r1-671b (Guo et al., 2025)	Open-weight	deepseek-chat
Llama-4-scout-17b-16e (Touvron et al., 2023)	Open-weight	meta-llama/llama-4-scout
Llama-4-maverick-17b-128e (Touvron et al., 2023)	Open-weight	meta-llama/llama-4-maverick
Qwen-max (Yang et al., 2024)	Proprietary	qwen-max
Qwen-plus (Yang et al., 2024)	Proprietary	qwen-plus-latest
Qwen3-235b-a22b (Yang et al., 2024)	Open-weight	qwen3-235b-a22b
Qwen3-32b (Yang et al., 2024)	Open-weight	qwen3-32b
QwQ-32b (QwenTeam, 2025)	Open-weight	qwq-32b
QwQ-plus (QwenTeam, 2025)	Proprietary	qwq-plus

943
944
A IMPLEMENTATION DETAILS
945946
A.1 DETAILS OF LLMs
947

948 Some key hyper-parameters of LLMs for results reported in baseline test and ORACLE benchmark
949 are set as follows: The temperature for all benchmarked models is set as 0. The reasoning ef-
950 fort for GPT-series LLMs (o1, o3-mini, o3, o4-mini) is set as medium. The token budget for
951 extended thinking in Claude-series LLMs (claude-3.7-sonnet_thinking, claude-4-sonnet_thinking)
952 is set as 20000. For Gemini-series LLMs (gemini-2.5-flash_thinking, gemini-2.5-pro), the think-
953 ing budget is set as dynamic. For Qwen-series LLMs (qwen-plus_thinking, qwen3-32b_thinking,
954 qwen3-235b-a22b_thinking, qwq-plus), the thinking budget is set as 20000 tokens. For deepseek-
955 r1, the length of thinking content can not be modified. So full thinking is allowed. The experi-
956 ments are conducted from 4 July to 19 Aug. All benchmarked models are up-to-date. For open-
957 weight LLMs like DeepSeek-series models, Llama-series models, Qwen-series models, we directly
958 call API from <https://api.deepseek.com>, <https://openrouter.ai/api/v1>,
959 <https://dashscope.aliyuncs.com/compatible-mode/v1> respectively.

960
A.2 DETAILS OF ORACLE BENCHMARK
961

962 To balance the cost of LLMs, the number of test samples for each black-box task is set as follows:

- 963 • Code Intent Inference (CII): The test samples include 5 unique input variable values, each
964 with 6 or 7 checkpoint questions, totaling 30 or 35 questions.
965
- 966 • Circuit Rule Inference (CRI): Each black-box considers 10 different input wires as test
967 samples.
968
- 969 • Physics System Inference (PSI): Each black-box considers 6 different time points as test
970 samples.
971
- 972 • Encryption Rule Inference (ERI): Each black-box considers 8 different plaintext as test
973 samples.

972 • Interactive Puzzle Inference (IPI): Each black-box considers 6 different puzzle answers as
 973 test samples.
 974
 975 • Game Strategy Inference (GSI): Each black-box considers 4 different game rounds, ranging
 976 from 8 to 15.
 977

978 Test samples are directly generated by LLMs. However, human rewriting is involved in some cases.
 979 We find LLMs fail to generate valid and good test samples in Code Intent Inference task. In Interac-
 980 tive Puzzle Inference, some test samples involve numerical calculations (e.g., Wordle), which Large
 981 Language Models (LLMs) sometimes struggle with.

982 B ADDITIONAL RELATED WORK

983 B.1 DATA CONTAMINATION AND DYNAMIC BENCHMARK

984 Previous works have highlighted the risks of memorization and contamination during LLM training
 985 and fine-tuning (Roberts et al., 2023; Deng et al., 2023): While LLMs are trained over a huge amount
 986 of data from Internet, static datasets will be inadvertently included, leading to overestimation of
 987 model performance (Magar & Schwartz, 2022; Balloccu et al., 2024). Therefore, researchers begin
 988 to shed light on dynamic benchmarks. Current approaches on building dynamic benchmarks can
 989 be classified into updating benchmark data based on the timestamps of LLM (White et al., 2024;
 990 Jain et al., 2024; Mahdavi et al., 2025) and regenerating benchmark data to reconstruct original
 991 benchmarks. Specifically, the latter approach can be further divided into rule-based reconstruction
 992 (Lei et al., 2023; Zhu et al., 2024; Mirzadeh et al., 2024; Zhao et al., 2024; Kurtic et al., 2024),
 993 LLM-based reconstruction (Ying et al., 2024; Cao et al., 2024; Qian et al., 2024), human-based
 994 reconstruction (Srivastava et al., 2024; Huang et al., 2025), and hybrid reconstruction (Zhang et al.,
 995 2024).
 996

997 B.2 EVALUATION OF REASONING ABILITY

1000 The growing complexity of reasoning tasks undertaken by LLMs makes their evaluation increas-
 1001 ingly difficult. Simply relying on the comparisons between LLM-generated outcomes and ground
 1002 truth labels (Cobbe et al., 2021; Hendrycks et al., 2021) becomes insufficient, as LLMs can produce
 1003 correct answers with logically incorrect reasoning process (Turpin et al., 2023; Hao et al., 2024;
 1004 Mondorf & Plank, 2024). Therefore, researchers turn to evaluate reasoning paths step-by-step. Ex-
 1005 isting criteria of metrics can be categorized into groundedness, validity, coherence, and utility (Lee
 1006 & Hockenmaier, 2025). Groundedness measures if the step is factually true according to the query
 1007 (Lewis et al., 2020). Validity evaluates if a reasoning step contains no errors (Lightman et al., 2023).
 1008 Coherence checks if the inputs for a reasoning step are adequately provided by the prior steps (Wang
 1009 et al., 2022). Utility measures if a reasoning step contributes to the correct final answer. Besides
 1010 leveraging evaluation metrics, LLM-as-a-judge is frequently employed in evaluation (Zheng et al.,
 1011 2023; Hao et al., 2024; Sun et al., 2024), which is a fast and cheap alternative to human judgment.
 1012

1013 B.3 CHARLES PEIRCE’S FRAMEWORK OF HUMANS REASONING BEHAVIOR

1014 The reasoning behavior of humans can be categorized into deductive reasoning, inductive reasoning,
 1015 and abductive reasoning according to Charles Peirce’s framework (Peirce, 1934). Generally speak-
 1016 ing, as shown in Figure 1, the reasoning process begins with abduction, where initial observations
 1017 spark potential explanatory hypotheses. These hypotheses are applied to derive new observations
 1018 via deduction. Then induction works in strengthening or discarding hypotheses by analyzing for-
 1019 mer and new observations. This cycle repeats, iteratively refining existing hypotheses until a robust
 1020 and generalizable theory is built (Burks, 1946; Fann, 2012). Charles Peirce’s framework reveals the
 1021 significance of human reasoning when facing an unknown environment: three aspects of reason-
 1022 ing are dynamically intertwined, collectively driving the discovery, verification, and application of
 1023 knowledge.
 1024

1026 **C ADDITIONAL DISCUSSION**
10271028 **C.1 LIMITATIONS**
10291030 The limitations of this work include: (i) This paper’s scope is limited to investigating the perfor-
1031 mance of LLMs in the ORACLE benchmark, with the evaluation of LLM-based agents reserved for
1032 future research. (ii) Due to the heavy cost of calling LLMs, we don’t evaluate some powerful yet
1033 expensive LLMs (e.g., o3-pro). The ORACLE benchmark also lacks a statistical analysis. (iii) Since
1034 the models that performed well on the ORACLE benchmark are all closed-source, we are unable to
1035 analyze them to determine why they fail to develop efficient and adaptive strategies in black-box
1036 interaction tasks. One possible explanation is that, it’s well-known that the capabilities of an LLM
1037 primarily depend on the base model’s pre-training and the post-training. The former involves next
1038 token prediction given a context, while the latter is an RL training on CoT prompt-based QA pairs.
1039 Neither of these training paradigms involves placing the model in an unknown environment for ex-
1040 ploration. Consequently, the model inherently lacks the ability for strategy optimization through
1041 multi-turn interaction with an environment.
10421043 **C.2 COMPARISON WITH PREVIOUS WORK**
10441045 Some previous researches (Wu et al., 2023; Hu et al., 2024; Shi et al., 2025) evaluate LLMs’ ability
1046 of playing text games. Black-box interaction differs from these works in three aspects: First, we
1047 focus on building an interactive, unknown environment with hidden rules and investigating LLMs
1048 behavior in this environment, instead of testing LLMs’ performance in playing specific games. Sec-
1049 ond, some of the text games can be viewed as a subset of Interactive Puzzle Inference in the ORACLE
1050 benchmark, and our proposed black-box interaction approach can easily scale the Interactive Puzzle
1051 Inference task. Third, all chosen text games in previous works are well-known, which increases the
1052 possibility that LLMs are already familiar with the exploration strategy. Black-boxes in Interactive
1053 Puzzle Inference are all modified to avoid this situation (detailed in Appendix G.5).
10541055 He et al. (2025) introduced RULEARN benchmark which builds an interactive environment with
1056 unknown rules. There exist three noticeable differences between RULEARN and the ORACLE
1057 benchmark. First, the ORACLE benchmark formalizes the black-box interaction task into distinct
1058 exploration and evaluation stages. In contrast, RULEARN only involves a one-stage multi-turn in-
1059 teraction, thus lacking an analysis of the reasons behind model failure. Second, the three tasks in
1060 RULEARN can be classified as Interactive Puzzle Inference (IPI) task within the ORACLE bench-
1061 mark. In other tasks within the ORACLE benchmark, the model must infer the hidden rule rather than
1062 a hidden answer, which is a more fundamental and challenging task. Furthermore, the black-box in-
1063 teraction concept we propose is highly flexible and generalizable. Its goal is to investigate whether
1064 a model can understand an unknown environment through exploration, thus accommodating a wide
1065 range of task settings beyond the six types in the ORACLE benchmark. In contrast, the task settings
1066 in RULEARN are singular and difficult to generalize.
10671068 **C.3 MORE TASK SETTINGS FOR BLACK-BOX INTERACTION**
10691070 In this section, we explore additional potential task settings for black-box interaction. The setting
1071 described in Section 2, which allows for test-time learning, is designed by considering possible
1072 human evaluation and shot numbers in evaluation metrics. A key distinction when involving human
1073 evaluators is the persistence of information. Unlike large language models (LLMs) that can easily
1074 delete messages from their dialogue history, humans retain previously seen content in their memory.
1075 Therefore, all prior test samples must remain part of the dialogue history. While a k -shot evaluation
1076 metric is necessary, we also inform the LLM whether its responses to test samples in its k trials were
1077 correct. More diverse task settings become feasible when human evaluation and shot numbers are
1078 not primary considerations. Their advantages and disadvantages are discussed below.
10791076 **C.3.1 FROM ALLOWING TEST-TIME EXPLORATION TO NOT ALLOWING**
10771078 Test-time exploration is forbidden, which means every test sample and LLMs’ answer will be re-
1079 moved from the dialogue history once it’s completed. LLMs only rely on information gained from
the exploration stage.

1080
 1081
 1082
 1083
 1084
 1085

- **Advantages:** Evaluation and exploration are totally disentangled, which is a clearer approach.
- **Disadvantages:** This setting is contradictory to human evaluation.

C.3.2 FROM TEST SAMPLES TO CODE EXPRESSION

Rather than directly providing answers for each test sample, models are now instructed to generate executable code that expresses their understanding of the black-box, which is then validated against test samples.

1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102

- **Advantages:** Code expression is a decent way to judge whether LLMs truly understand the black-box rather than simply pattern matching, which is more fundamental to the core concept of black-box interaction. More importantly, code allows for checking millions of test samples quickly and is especially useful for some specific black-boxes. For examples, in some black-boxes in GSI task that involves a random opponent strategy, code expression is capable of simulating millions of games to judge whether models truly understand opponent’s strategy. In PSI task, code expression is effective for complicated motion without analytical solution . We already apply code expression for “Double Pendulum”, “Harmonic with Friction”, “Ball Air Resistance” black-boxes in the PSI task (detailed in Appendix G.3).
- **Disadvantages:** This approach introduce the additional challenge of coding ability of LLMs, which contradicts to the original of building a pure reasoning benchmark. As the possibility of a model can answer but fail to write correct code exists. In this setting, k -shot evaluation is also inapplicable.

C.3.3 FROM FULLY OBSERVABLE TO PARTIALLY OBSERVABLE

In current setting, black-box returns complete state information to models’ query, which makes up of fully observable black-box interaction. Partially observable black-box interaction only returns part of black-box information. For example, black-box will only returns values of a subset of variables in CII task. Partially observable black-box interaction is a harder version of fully observable black-box interaction.

1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117

- **Advantages:** Partially observable black-box interaction generalizes the current setting and better reflect real-world scenarios. It further challenges models’ advanced reasoning by increasing the difficulty of aggregating imperfect information. Exploration turns are also supposed to be extended, which tests models’ long-context reasoning.
- **Disadvantages:** Not applicable.

C.3.4 FROM FIXED EXPLORATION TURNS TO DYNAMIC

The goal of models is accurately answer test samples with the fewest possible exploration turns, rather than given fixed exploration turns.

1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125

- **Advantages:** This setting significantly tests a model’s planning capabilities, requiring the development of highly effective exploration strategies for strong performance.
- **Disadvantages:** When exploration strategies become dynamic, comparing model performance gets tricky. It’s tough to decide if a model with less exploration but lower accuracy is better than one with more exploration and higher accuracy.

C.4 EXTRA FINDINGS

1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133

Some extra findings during experiments are reported here. First, we find some LLMs, especially gemini-2.0-flash and gemini-2.5-flash, perform bad in instruction following. Black-box interaction requires accurate output format. While LLMs are given chances to correct formatting mistakes, continued disobedience of the specified format results in an invalid interaction turn. Second, the time cost of black-box interaction is an issue worthy of attention. We find that o4-mini and gemini-2.5-pro achieve the best balance between accuracy and time cost, while the time cost of qwen-series models and deepseek-r1 is extremely high.

We have also identified three additional weaknesses of LLMs in tackling black-box interaction tasks. First, LLMs primarily rely on pattern matching to understand black-box. Prior knowledge is essential for hypothesis developing and verifying in abductive and inductive reasoning. However, we find that LLMs rely heavily on prior knowledge and matching black-box function to familiar patterns, rather than engaging in genuine exploration. This phenomenon is most evident in the CII task where all black-boxes are famous coding algorithms. LLMs can quickly identify the hidden algorithm with only few observations over checkpoint output. So despite the difficult setting of CII, LLMs still perform well. But when it comes to PSI task where a black-box is a free combination of moving objects, or ERI task where the black-boxes are variations of well-known encryption algorithms, the performance of LLMs becomes relatively low. Notably, almost all models (including o3, gemini-2.5-pro) fail to beat a simple black-box that plays rock and scissors with equal probability in GSI 2@1 (as shown in Figure 2 (f)). Second, LLMs struggle in reasoning over dense information. LLMs are supposed to spend more turns for exploration when the black-box function is complex. They cannot achieve good performance without the ability to reason over dense information. This weakness is most evident in “Wordle” and “Quordle” black-boxes in the IPI task. LLMs can easily guess a 11-letter word in “Wordle” within 10 rounds, but fail to guess four 8-letter words in “Quordle” within 20 rounds. Third, even best-performing reasoning LLMs fall short in basic computing ability. For example, in a black-box implementing simple harmonic motion in PSI task, gemini-2.5-pro successfully identify the motion behavior but fail in correctly calculating the coordinates. Another example is the “Nerdle” black-box in IPI task which requires LLMs to output a 15-character equation. Most models fail to calculate if the output only contains 15 characters.

D CASE STUDY

D.1 HOW ITERATIVE DEBUGGING WORKS

The effectiveness of our iterative debugging framework stems from its ability to uncover a wide range of errors that are often missed in a single-pass generation process. By simulating a full interaction—akin to a human learning a game by playing it—the framework can identify and rectify several classes of bugs that a programmer might make. These include:

1. **Violations of unstated “common-sense” rules:** Task descriptions often omit implicit constraints, such as the fact that a player’s score or money cannot be negative. Our interactive process makes these violations apparent, forcing a correction.
2. **Misinterpretations of ambiguous language:** Natural language can be imprecise. The framework corrects for misunderstandings, as demonstrated in the example of circuit task where the term “random” was initially misinterpreted, leading to a non-deterministic implementation instead of a fixed random one.
3. **Simple yet critical implementation bugs:** This category includes flaws analogous to typos or logical oversights, such as using an incorrect formula in the example of physics task. It also includes bugs that cause runtime errors. These are difficult to spot in a static code review but are readily exposed when the simulation produces incorrect outputs.

Therefore, our framework significantly lowers the natural language description requirements for platform development. It tolerates ambiguity and allows for the omission of details, even granting the coding agent the freedom to elaborate on aspects that don’t compromise the platform’s core functionality. Furthermore, it enhances accuracy by autonomously correcting runtime and other logical errors. This substantially lowers the costs associated with benchmark construction, fine-tuning, and scalability.

The following case studies provide concrete examples of how our framework addresses flaws to produce robust and correct platform code.

1188 **An Example in Circuit Rule Inference Task** In this example from the Circuit Rule Inference
 1189 Task, the initial natural language description provided to the Coding LLM was: "For input size n=4,
 1190 the blackbox contains a random circuit of 8 gates. The circuit is not for some well-known purpose."
 1191 The Coding LLM misinterpreted the term "random." While the circuit's structure was meant to be
 1192 random, it needed to be fixed throughout a single interaction session to represent a deterministic
 1193 physical device. Instead, the initial code 1 regenerated a new random circuit for every call to the
 1194 blackbox function. As shown in Figure 11a and Figure 12, the Test LLM's interaction quickly
 1195 exposed this flaw by providing identical inputs at different rounds and receiving different outputs, a
 1196 logical contradiction that the Refinement LLM then used to correct the code 2.
 1197

```

1198   1 # Conical Pendulum Version 1.0
1199   2 def blackbox(circuit_input):
1200   3     """
1201   4       Simulates a random boolean circuit with
1202   5           ↪ 4 input wires and 8 gates.
1203   6           Returns the output of each gate as a
1204   7           ↪ list of 0/1 bits.
1205   8
1206   9           n = 4 # number of input wires
1207  10          m = 8 # number of gates
1208  11
1209  12          # Randomly construct a circuit, each
1210  13          ↪ gate's inputs come from input wires
1211  14          ↪ or previous gates
1212  15          gates = []
1213  16          for i in range(m):
1214  17              gate_type = random.choice(['AND',
1215  18                  ↪ 'OR', 'NOT'])
1216  19              # For gate i (indexing from 0),
1217  20              # possible sources: input wires
1218  21              # (1..n) or previous gates (1..i)
1219  22              inputs = []
1220  23              for idx in range(1, n+1):
1221  24                  inputs.append( (0, idx) )
1222  25              for idx in range(1, i+1):
1223  26                  inputs.append( (1, idx) )
1224  27              if gate_type in ['AND', 'OR']:
1225  28                  # Randomly select two inputs
1226  29                  choices = random.sample(inputs,
1227  30                      ↪ 2)
1228  31                  input1, input2 = choices[0],
1229  32                  ↪ choices[1]
1230  33                  gates.append( (gate_type,
1231  34                      ↪ input1, input2) )
1232  35              elif gate_type == 'NOT':
1233  36                  # Need one input
1234  37                  input1 = random.choice(inputs)
1235  38                  gates.append( (gate_type,
1236  39                      ↪ input1) )
1237  40
1238  41          # Simulate the circuit
1239  42          result = simulate_circuit(n, m,
1240  43              ↪ circuit_input, gates)
1241  44
1242  45          return result
  
```

1230 **Listing 1: Code of v1 platform**
 1231 (randomly generate circuit repeatedly)

1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241

```

1 # Conical Pendulum Version 2.0
2 # Circuit parameters
3 N = 4 # number of input wires
4 M = 8 # number of gates
5
6 # Generate the circuit once per run to
7 # ensure deterministic behavior for the
8 # same input
9 _CIRCUIT_GATES = []
10 for i in range(M):
11     gate_type = random.choice(['AND', 'OR',
12         ↪ 'NOT'])
13     # possible sources: input wires
14     # (0,1..N) or previous gates (1..i)
15     inputs = [(0, idx) for idx in range(1,
16         ↪ N + 1)]
17     for idx in range(1, i + 1):
18         inputs.append((1, idx))
19     if gate_type in ['AND', 'OR']:
20         if len(inputs) < 2:
21             input1, input2 = (0, 1), (0, 2)
22         else:
23             input1, input2 =
24                 ↪ random.sample(inputs, 2)
25         _CIRCUIT_GATES.append((gate_type,
26             ↪ input1, input2))
27     else: # NOT
28         input1 = random.choice(inputs)
29         _CIRCUIT_GATES.append((gate_type,
30             ↪ input1))
31
32 def blackbox(circuit_input):
33     """
34     Simulates the fixed random boolean
35     circuit with 4 input wires and 8
36     gates.
37     Returns the output of each gate as a
38     ↪ list of 0/1 bits.
39     """
40
41     # Simulate the circuit
42     result = simulate_circuit(N, M,
43         ↪ circuit_input, _CIRCUIT_GATES)
44
45     return result
  
```

1232 **Listing 2: Code of v2 platform**
 1233 (generate circuit only once)

1242
 1243 (...Instructions Omitted...)

1244 user: Welcome to the Boolean Circuit Game!
 1245 You are interacting with a blackbox circuit with
 1246 4 input wires and 8 logic gates. (...) [Round 2/20
 1247 | 19 rounds left]

1248 LLM: (0, 0, 0, 1)

1249 user: Gate outputs for your input [0, 0, 0,
 1250 1]: 1 1 0 1 1 1 1 1 Please submit your next input
 1251 bits for round 3. (...) [Round 10/20 | 11 rounds
 1252 left]

1253 LLM: (1, 0, 0, 1)

1254 user: Gate outputs for your input [1, 0, 0,
 1255 1]: 1 0 1 0 1 0 0 0 Please submit your next input
 1256 bits for round 11. (...) [Round 17/20 | 4 rounds
 1257 left]

1258 LLM: (0, 0, 0, 1)

1259 user: Gate outputs for your input [0, 0, 0,
 1260 1]: 1 1 1 1 1 0 0 1 Please submit your next input
 1261 bits for round 18. [Round 18/20 | 3 rounds left]

1262 LLM: (1, 0, 0, 1)

1263 user: Gate outputs for your input [1, 0, 0,
 1264 1]: 0 0 0 0 0 1 1 0 (...)

1265 *****Evaluation Starts, You Have 1
 1266 Chances for Answering Each Question*****
 1267 The output format is described in the Evaluation
 1268 section previosly. For example: [0, 1, 0, 1] In
 1269 this turn, given the input [0, 0, 0, 0], answer the
 1270 output of the gates in the format we dicussed
 1271 without any text else.

1272 LLM: [0, 0, 1, 0, 1, 0, 0, 0]

1273 user: the answer is wrong when input = [0,
 1274 0, 0, 0] Let's move to next question.

1275 LLM: Ok.

1282 (...Instruction Omitted...)

1283 user: Welcome to the Boolean Circuit Game!
 1284 You are interacting with a blackbox circuit with
 1285 4 input wires and 8 logic gates. (...) [Round 2/20
 1286 | 19 rounds left]

1287 LLM: (0, 0, 0, 1)

1288 user: Gate outputs for your input [0, 0, 0,
 1289 1]: 0 1 0 0 0 1 0 1 Please submit your next input
 1290 bits for round 3. (...) [Round 10/20 | 11 rounds
 1291 left]

1292 LLM: (1, 0, 0, 1)

1293 user: Gate outputs for your input [1, 0, 0,
 1294 1]: 1 0 1 1 0 0 1 0 Please submit your next input
 1295 bits for round 11. (...) [Round 17/20 | 4 rounds
 1296 left]

1297 LLM: (0, 0, 0, 1)

1298 user: Gate outputs for your input [0, 0, 0,
 1299 1]: 0 1 0 0 0 1 0 1 Please submit your next input
 1300 bits for round 18. [Round 18/20 | 3 rounds left]

1301 LLM: (1, 0, 0, 1)

1302 user: Gate outputs for your input [1, 0, 0,
 1303 1]: 1 0 1 1 0 0 1 0 (...)

1304 *****Evaluation Starts, You Have 1
 1305 Chances for Answering Each Question*****
 1306 The output format is described in the Evaluation
 1307 section previosly. For example: [0, 1, 0, 1] In
 1308 this turn, given the input [0, 0, 0, 0], answer the
 1309 output of the gates in the format we dicussed
 1310 without any text else.

1311 LLM: [0, 1, 0, 0, 1, 1, 0, 1]

1312 user: the answer is wrong when input = [0,
 1313 0, 0, 0] Let's move to next question.

1314 LLM: Ok.

(a) Interaction log (v1 platform)

(b) Interaction log (v2 platform)

Figure 11: Black-box interaction log generated using different platform code

1315 - The same input [0, 0, 0, 1] produced two different outputs:

1316 • Round 2 → 1 1 0 1 1 1 1 1

1317 • Round 17 → 1 1 1 1 1 0 0 1

1318 - The same input [1, 0, 0, 1] produced two different outputs:

1319 • Round 10 → 1 0 1 0 1 0 0 0

1320 • Round 18 → 0 0 0 0 0 1 1 0

1321 These contradictions violate the requirement that a fixed combinational circuit must give the same out-
 1322 puts for identical inputs.

Figure 12: Logical error figured out by Refinement LLM

1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349

An Example in Physics System Inference Task This case demonstrates the framework’s ability to detect and correct subtle but critical bugs in the implementation logic, corresponding to the third type of error mentioned above. The task is to simulate a conical pendulum, a standard physics problem. The initial code generated by the Coding LLM 3 contained a flaw in the physical formula for calculating the angular velocity ω , incorrectly using $\cos(\theta)$ where $\tan(\theta)$ was required. This type of error is analogous to a small logical oversight by a human programmer—it is syntactically correct but semantically wrong, making it difficult to catch without executing the code and validating its output. During the simulation phase 13a, the framework discovered that the coordinates produced by the v1 platform were inconsistent with the expected physical behavior, leading to failed evaluation checks in 14. The Refinement LLM then correctly identified the erroneous formula as the root cause and produced the corrected code 4.

```

1 # Conical Pendulum Version 1.0
2 def blackbox(t):
3     # Conical pendulum parameters
4     length = 5 # m
5     theta = math.radians(30) # 30 degrees
6     ↪ in radians
7     g = 10 # m/s2
8
9     # Calculate motion parameters
10    omega = math.sqrt(g * math.cos(theta)) /
11        ↪ length) # angular velocity
12    radius = length * math.sin(theta) #
13    ↪ radius of circular motion
14    z_position = -length * math.cos(theta)
15    ↪ # constant z position (negative,
16    ↪ below origin)
17
18    # Calculate 3D coordinates
19    x = radius * math.cos(omega * t)
20    y = radius * math.sin(omega * t)
21    z = z_position
22
23    # Round to two decimal places
24    x = round(x, 2)
25    y = round(y, 2)
26    z = round(z, 2)
27
28    object_coordinate = {"object1": (x, y,
29    ↪ z)}
30    return object_coordinate

```

Listing 3: Code of v1 platform

```

1 # Conical Pendulum Version 2.0
2 def blackbox(t):
3     # Conical pendulum parameters
4     length = 5 # m
5     theta = math.radians(30) # 30 degrees
6     ↪ in radians
7     g = 10 # m/s2
8
9     # Calculate motion parameters
10    omega = math.sqrt(g * math.tan(theta)) /
11        ↪ length) # angular velocity
12    radius = length * math.sin(theta) #
13    ↪ radius of circular motion
14    z_position = -length * math.cos(theta)
15    ↪ # constant z position (negative,
16    ↪ below origin)
17
18    # Calculate 3D coordinates
19    x = radius * math.cos(omega * t)
20    y = radius * math.sin(omega * t)
21    z = z_position
22
23    # Round to two decimal places
24    x = round(x, 2)
25    y = round(y, 2)
26    z = round(z, 2)
27
28    object_coordinate = {"object1": (x, y,
29    ↪ z)}
30    return object_coordinate

```

Listing 4: Code of v2 platform

1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378

(...Instructions Omitted...)
 Black-box: You have 6 interaction turns to understand the black-box. Now the interaction starts. Only output the value and DO NOT contain any unrelated text.
 LLM: 0
 Black-box: (Current Turn: 1, 5 Turns Remaining)
 {‘object1’: (2.5, 0.0, -4.33)}
 LLM: 1
 Black-box: (Current Turn: 2, 4 Turns Remaining)
 {‘object1’: (0.63, 2.42, -4.33)}
 LLM: 2
 Black-box: (Current Turn: 3, 3 Turns Remaining)
 {‘object1’: (-2.18, 1.22, -4.33)}
 LLM: 3
 Black-box: (Current Turn: 4, 2 Turns Remaining)
 {‘object1’: (-1.73, -1.8, -4.33)}
 LLM: 4
 Black-box: (Current Turn: 5, 1 Turns Remaining)
 {‘object1’: (1.31, -2.13, -4.33)}
 LLM: 5
 Black-box: (Current Turn: 6, 0 Turns Remaining)
 {‘object1’: (2.39, 0.73, -4.33)} *****Evaluation Starts, You Have 1 Chances for Answering Each Question***** Now answer the question: What is the coordinate of each object at time 1.5?
 LLM: {‘object1’: (-0.78, 2.29, -4.33)}
 Black-box: Your answer is wrong. Let’s move to next question.
 LLM: Ok.

1379
 1380

(a) Interaction log (v1 platform)

1381
 1382
 1383
 1384
 1385
 1386

(...Instructions Omitted...)
 Black-box: You have 6 interaction turns to understand the black-box. Now the interaction starts. Only output the value and DO NOT contain any unrelated text.
 LLM: 0
 Black-box: (Current Turn: 1, 5 Turns Remaining)
 {‘object1’: (2.5, 0.0, -4.33)}
 LLM: 1
 Black-box: (Current Turn: 2, 4 Turns Remaining)
 {‘object1’: (1.19, 2.2, -4.33)}
 LLM: 2
 Black-box: (Current Turn: 3, 3 Turns Remaining)
 {‘object1’: (-1.37, 2.09, -4.33)}
 LLM: 3
 Black-box: (Current Turn: 4, 2 Turns Remaining)
 {‘object1’: (-2.49, -0.21, -4.33)}
 LLM: 4
 Black-box: (Current Turn: 5, 1 Turns Remaining)
 {‘object1’: (-1.01, -2.29, -4.33)}
 LLM: 5
 Black-box: (Current Turn: 6, 0 Turns Remaining)
 {‘object1’: (1.53, -1.97, -4.33)} *****Evaluation Starts, You Have 1 Chances for Answering Each Question***** Now answer the question: What is the coordinate of each object at time 1.5?
 LLM: {‘object1’: (0.11, 2.22, -4.33)}
 Black-box: Your answer is wrong. Let’s move to next question.
 LLM: Ok.

(b) Interaction log (v2 platform)

Figure 13: Black-box interaction log generated using different platform code

1400
 1401
 1402
 1403

1387
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403

Looking at the interaction log carefully:
 Wrong points identified:
 1. **Insufficient analysis of the mechanical system**: The assistant fails to properly analyze the motion pattern. From the coordinates provided:
 - t=0: (2.5, 0.0, -4.33)
 - t=1: (0.63, 2.42, -4.33)
 - t=2: (-2.18, 1.22, -4.33)
 - t=3: (-1.73, -1.8, -4.33)
 - t=4: (1.31, -2.13, -4.33)
 - t=5: (2.39, 0.73, -4.33)
 The z-coordinate remains constant at -4.33, and the x,y coordinates trace a circular path, indicating conical pendulum motion. However, the assistant doesn’t demonstrate understanding of this pattern in their final prediction.

Figure 14: Logical error figured out by Refinement LLM

1404
1405

D.2 HOW LLMs INTERACT WITH THE BLACK-BOX

1406

An Example in Interactive Puzzle Inference Task

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

developer: You are an excellent reasoning model. You are very good at understanding the interaction rules, reasoning over interaction history, thinking step by step, and finally figuring out the working principle of a system that contains hidden information.

You also strictly follow output format rules. When making queries, you strictly obey querying format. When answering questions, you strictly obey answer format. You never output anymore unrelated text.

user: 1. Task overview:

- The user plays the role of a puzzle, and you don't know what the hidden answer is. You need to guess the hidden answer by interacting with the user in multiple turns.

2. Goals:

- You need to guess the answer to the puzzle within given interaction turns.

3. User property:

- The user hides the answer which you need to figure out.

4. Interaction rules:

- Rule 0: The user will first tell you the rule of the puzzle, and the interaction format that must be followed when querying. In each turn, the user will tell *current turn* and *remaining turns*.

- Rule 1: You can ask questions according to the rules of the game and receive corresponding feedback. If your ask is unavailable, the user will tell you.

- Rule 2: After a series of interactions, you should answer the puzzle in the format specified in the description.

5. Output format:

- When you ask a question, you should strictly follow query format in the **Description**.

- When you answer the puzzle, you should strictly follow the answer format in the **Description**.

- Refer to the examples in the **Description**. for the correct format.

- If you figure out the right answer before given turns, keep interacting with the puzzle to make sure your answer is correct.

6. Evaluation:

- When the given number of interactions is reached, you need to give your answer of the puzzle. **You MUST ONLY output the answer itself in the format mentioned in the description, DO NOT contain more text.**

Now Let's Solve the Puzzle wordle.

Description: Wordle is a word-guessing game where players attempt to deduce a hidden 8-letter word (all uppercase). Each guess provides feedback (1) Correct letter in the correct position, represented by 'A'; (2) Correct letter but misplaced, represented by 'M'; (3) Letter not in the word, represented by 'X'.

Players iteratively refine guesses using feedbacks until solving the word, typically within limited attempts. After a few times of querying, the player will give an 8-letter uppercase word answer, and he'll win if he answers correctly.

OUTPUT the 8-letter uppercase word directly.

LLM: I understand the rules. I will not output any unrelated text! Let us start the interaction.

user: *****A New Puzzle Starts, You can Make 10 Queries Before Answering Each Question. And Then You Have 1 Chances for Answering. Output the Value Only.*****

LLM: AIRPLANE

user: <Current Turn: 1, 9 Turns Remaining> AXMXMMMM

LLM: ABALONES

user: <Current Turn: 2, 8 Turns Remaining> AXMMXMMX

LLM: ARLENDTC

user: <Current Turn: 3, 7 Turns Remaining> AAMMAXXM

LLM: ARCANELY

1458
 1459 user: <Current Turn: 4, 6 Turns Remaining> AAAAAAAA
 1460
 1461 LLM: ARCANELY
 1462
 1463 user: <Current Turn: 5, 5 Turns Remaining> AAAAAAAA
 1464
 1465 LLM: ARCANELY
 1466
 1467 user: <Current Turn: 6, 4 Turns Remaining> AAAAAAAA
 1468
 1469 LLM: ARCANELY
 1470
 1471 user: <Current Turn: 7, 3 Turns Remaining> AAAAAAAA
 1472
 1473 LLM: ARCANELY
 1474
 1475 user: <Current Turn: 8, 2 Turns Remaining> AAAAAAAA
 1476
 1477 LLM: ARCANELY
 1478
 1479 user: <Current Turn: 9, 1 Turns Remaining> AAAAAAAA
 1480
 1481 LLM: ARCANELY
 1482
 1483 user: Your answer is correct.
 1484
 1485 LLM: Ok.

1488 An Example in Game Strategy Inference Task

1489
 1490 developer: You are an excellent reasoning model. You are very good at understanding the interaction
 1491 rules with a black-box, reasoning over the interaction history, thinking step by step, and finally guessing
 1492 the pattern of the black-box and finding a way to beat it in a game. You also strictly follow output format
 1493 rules.
 1494 user: 1. Task overview: - The user plays the role of an opponent who takes a fixed strategy in a
 1495 game. But you don't know what the strategy is. You need to guess the hidden strategy of your opponent
 1496 by interacting (playing game) with him in multiple turns.
 1497 2. Goals: - You have 1 final goal: You need to guess your opponent's strategy and try to maximize your
 1498 score in the game. The score might depend on winning rate, or minimal cost, etc.
 1499 3. User property: - The user hides his game strategy which you need to figure out to win the game.
 1500 4. Interaction rules: - To finish the goal, you need to interact with the user. The interaction rules are as
 1501 follows: - Interaction Rule 0: The user will first tell you the rule of the game, and the interaction format
 1502 that must be followed when playing. In each turn, the user will tell *current turn* and *remaining turns*.
 1503 - Interaction Rule 1: You can take actions according to the rules of the game and receive corresponding
 1504 feedback, such as current game states. If your action is unavailable, the user will tell you. - Interaction
 1505 Rule 2: We will first play a few times of the game to familiarize you with the rules and the behavior of
 1506 your opponent. In this phase, your actions will not be recorded, and your score does not matter. You can
 1507 make use of this phase to explore the game and understand the opponent's strategy. - Interaction Rule 3:
 1508 After the *exploration phase*, you will enter the *evaluation phase*. We will only play the game for 1
 1509 final time, and your actions will be recorded. Your score will be calculated based on your actions in this
 1510 final game.
 1511 5. Output format: - **You must strictly obey the output format rules, DO NOT output any unrelated
 1512 text!**:
 6. Evaluation: - When the given number of interactions is reached, the game ends and we'll calculate
 1513 your **score**

1512
 1513 Now Let's Play the Game Load Shoot Defend Scout, the Description Is that In each turn, a player can
 1514 choose 'load': gain one bullet, 'scout': see the opponent's bullet count, 'shoot x': spend 'x' bullets to
 1515 attack, or 'defend y': spend 'y' bullets to defend. Then, actions and the gaining points of both players
 1516 are revealed, but the specific numerical values 'x' and 'y' are kept hidden. Each player starts with
 1517 0 bullets and can hold a maximum of 8. 'x' and 'y' cannot exceed the number of bullets they have
 1518 currently. If one player 'shoot x' and the other 'defend y', the defense succeeds and the player wins 1
 1519 point if $y \geq x$, and the shooter wins -1 point. But if $y < x$, the attacker wins 1 point and the defender
 1520 wins -1 point. A Shoot action will always succeed against an opponent who chooses to Load or Scout.
 1521 When a simultaneous shoot happens, player with more bullets will win 1 point and the other win -1
 1522 point. The same bullets will result in a tie where both players win 0 point. When a simultaneous defend
 1523 or scout or load happens, both players win 0 point. If player 'scout', he will receive opponent's bullet
 1524 number. The game lasts for [[total_turns]] turns, both players try to maximize their scores.
 1525
 1526 LLM: I understand the rules. I will not output any unrelated text! Let us start the interaction.
 1527
 1528 user: *****Exploration Phase Starts, We will Play the Game for 1 Times. Your Actions Will
 1529 Not Be Recorded, and Your Score Does Not Matter.*****
 1530
 1531 LLM: Please provide the game rules and the required interaction format so that I can take my
 1532 first action in the exploration phase.
 1533
 1534 user: ***Exploration Round <1/1> Start***
 1535
 1536 LLM: Ok. I'm ready to play the game. This is round 1 of the exploration phase.
 1537
 1538 user: Turn 1/8
 1539 This is the first turn.
 1540 You have 0 bullets.
 1541 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1542
 1543 LLM: load
 1544
 1545 user: Turn 2/8
 1546 In the last turn, you chose 'load' and the opponent chose 'load'. You gained 0 point(s).
 1547 You have 1 bullets.
 1548 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1549
 1550 LLM: I need the complete rules and scoring details to make informed choices. Could you please
 1551 provide the game's description, the available actions, and how points are awarded?
 1552
 1553 user: Invalid action type. Your action 'I need the complete rules and scoring details to make in-
 1554 formed choices. Could you please provide the game's description, the available actions, and how points
 1555 are awarded?' is not recognized. Please choose from 'load', 'scout', 'shoot x', or 'defend y'.
 1556
 1557 LLM: scout
 1558
 1559 user: Turn 3/8
 1560 In the last turn, you chose 'scout' and the opponent chose 'load'. You gained 0 point(s).
 1561 You have 1 bullets.
 1562 Your scout last turn revealed that the opponent had 2 bullets before their action.
 1563 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1564
 1565 LLM: scout
 1566
 1567 user: Turn 4/8
 1568 In the last turn, you chose 'scout' and the opponent chose 'shoot'. You gained 0 point(s).
 1569 You have 1 bullets.
 1570 Your scout last turn revealed that the opponent had 2 bullets before their action.
 1571 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1572
 1573 LLM: shoot 1
 1574
 1575 user: Turn 5/8

1566
 1567 In the last turn, you chose 'shoot 1' and the opponent chose 'load'. You gained 1 point(s).
 1568 You have 0 bullets.
 1569 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1570
 1571 LLM: load
 1572
 1573 user: Turn 6/8
 1574 In the last turn, you chose 'load' and the opponent chose 'shoot'. You gained 0 point(s).
 1575 You have 1 bullets.
 1576 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1577
 1578 LLM: scout
 1579
 1580 user: Turn 7/8
 1581 In the last turn, you chose 'scout' and the opponent chose 'load'. You gained 0 point(s).
 1582 You have 1 bullets.
 1583 Your scout last turn revealed that the opponent had 1 bullets before their action.
 1584 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1585
 1586 LLM: scout
 1587
 1588 user: Turn 8/8
 1589 In the last turn, you chose 'scout' and the opponent chose 'load'. You gained 0 point(s). You have 1
 1590 bullets. Your scout last turn revealed that the opponent had 2 bullets before their action. What is your
 1591 action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1592
 1593 LLM: defend 1
 1594
 1595 user: *****Evaluation Phase Starts, We Will Play the Game for 1 Time. Now is the 0 time.
 1596 The highest score Will Be Recorded.*****
 1597
 1598 LLM: I am ready
 1599
 1600 user: Turn 1/8
 1601 This is the first turn. You have 0 bullets.
 1602 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1603
 1604 LLM: load
 1605
 1606 user: Turn 2/8
 1607 In the last turn, you chose 'load' and the opponent chose 'load'. You gained 0 point(s). You have 1
 1608 bullets.
 1609 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1610
 1611 LLM: shoot 1
 1612
 1613 user: Turn 3/8
 1614 In the last turn, you chose 'shoot 1' and the opponent chose 'load'. You gained 1 point(s). You have 0
 1615 bullets.
 1616 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1617
 1618 LLM: load
 1619
 1620 user: Turn 4/8
 1621 In the last turn, you chose 'load' and the opponent chose 'shoot'. You gained 0 point(s). You have 1
 1622 bullets.
 1623 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1624
 1625 LLM: shoot 1
 1626
 1627 user: Turn 5/8
 1628 In the last turn, you chose 'shoot 1' and the opponent chose 'load'. You gained 1 point(s). You have 0
 1629 bullets.

1620
 1621 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1622
 1623 LLM: load
 1624
 1625 user: Turn 6/8
 1626 In the last turn, you chose 'load' and the opponent chose 'shoot'. You gained 0 point(s). You have 1
 1627 bullets.
 1628 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1629
 1630 LLM: shoot 1
 1631
 1632 user: Turn 7/8
 1633 In the last turn, you chose 'shoot 1' and the opponent chose 'load'. You gained 1 point(s). You have 0
 1634 bullets.
 1635 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1636
 1637 LLM: load
 1638
 1639 user: Turn 8/8
 1640 In the last turn, you chose 'load' and the opponent chose 'load'. You gained 0 point(s). You have 1
 1641 bullets.
 1642 What is your action? (e.g., 'load', 'scout', 'shoot 1', 'defend 2')
 1643
 1644 LLM: shoot 1

D.3 HOW LLMs SUCCEED AND FAIL

1644 **A Successful Example in Code Intent Inference Task** In this case, the black-box code implemented QuickSort, selecting the last element of the interval as the pivot. The LLM quickly identified this as QuickSort and correctly determined the pivot element.

1648 Black-Box: (Instructions omitted) <Current Turn: 1, 19 Turns Remaining> The black-box takes
 1649 ['name': 'arr', 'type': <class 'list'>] as input variables, and has 3 checkpoints.
 1650
 1651 Gemini-2.5-pro: <think>**Deconstructing a Function: My Initial Approach**Alright, I'm tasked with
 1652 understanding this function that takes a list, 'arr', as input and hits three checkpoints along the way. My
 1653 gut tells me the best strategy here is to start simple and build complexity. So, I'm thinking, let's feed it
 1654 a very straightforward input: a sorted list. I'm imagining 'arr' will start as '[1, 2, 3, 4, 5]'. This should
 1655 allow me to focus on the core logic without any confusing edge cases initially. My immediate next step
 1656 is to get a handle on what's happening at the first checkpoint, specifically at iteration '(1, 1)'. That's
 1657 where I need to start. I'm hoping this will give me a clear picture of the initial setup or the state at the
 1658 very beginning of any iterative process the function might employ. Knowing this initial state will be key
 1659 to understanding how the function progresses and eventually lands at those critical checkpoints. So I
 1660 need to query this state and observe what's happening with the input array at that initial point.</think>
 arr = [1, 2, 3, 4, 5] (1, 1)
 1661
 1662 Black-Box: <Current Turn: 2, 18 Turns Remaining> Error: Failed to convert [1, 2, 3, 4, 5](1,
 1) to list: invalid syntax (<string>, line 2)
 1663
 1664 Gemini-2.5-pro: <think>**Understanding the Input and Refining My Strategy**Okay, so I got
 1665 an error message back after my last attempt, indicating a problem with the parsing of my input. It
 1666 looks like my combined assignment and checkpoint request in a single command, 'arr = [1, 2, 3, 4, 5]'
 1667 and '(1, 1)', threw the system off. The "invalid syntax" error is a clear indicator that the system can't
 1668 handle them together. Rereading the rules, I realize I need to be more deliberate about the order of
 1669 operations and the output format. Rule 4 emphasizes that I need to handle variable assignments (Rule
 1670 1) *before* requesting checkpoint values (Rule 2). While the rules don't explicitly mandate separate
 1671 turns, the error suggests it's the safest and most reliable approach. Therefore, for this round, I'll focus
 1672 *solely* on assigning a value to 'arr'. I'll keep the input simple and valid – 'arr = [10, 20, 30]'. This
 1673 should ensure the system receives a clean and understandable variable assignment. The next turn will be
 1674 devoted to requesting checkpoint values using the appropriate format. This should minimize the chance
 1675 of triggering those parsing issues again.</think> arr = [10, 20, 30]

1674
 1675 Black-Box: <Current Turn: 3, 17 Turns Remaining> Set 'arr': [10, 20, 30].
 1676
 1677 Gemini-2.5-pro: <think> </think>
 1678
 1679 Black-Box: (Instructions omitted) <Current Turn: 1, 19 Turns Remaining> The black-box takes
 1680 ['name': 'num', 'type': list[int]] as input variables, and has 5 checkpoints.
 1681
 1682 Gemini-2.5-pro: <think>**Analyzing the Function's Behavior**Okay, so I'm given an array,
 1683 'arr', and the number of checkpoints, which is 3 in this case. My immediate task is to grasp the
 1684 function's internal workings. I've mentally initialized 'arr' as [10, 20, 30] to have some concrete data
 1685 to work with. Now, the plan is to use checkpoints to reconstruct the function's execution flow. By
 1686 strategically placing probes, I can see what variables are accessible and how their values evolve. The
 1687 first step is checking the first checkpoint. Since the checkpoints are numbered, this likely corresponds
 1688 to the beginning of the function's core loop or main logic. Probing at (1, 1) – the first checkpoint of the
 1689 first iteration – will give me the initial state. I anticipate seeing 'arr', as it's the input. There may also be
 1690 some initial loop counter(s) or any initial state variables. This will lay the groundwork for understanding
 1691 the process.</think> (1, 1)
 1692
 1693 Black-Box: <Current Turn: 4, 16 Turns Remaining> ['name=arr, value=[10, 20, 30], type=list',
 1694 'name=p, value=30, type=int']
 1695
 1696 Gemini-2.5-pro: <think>**Initial Analysis of the Hidden Code Function**Okay, so I'm pre-
 1697 sented with a function and my job is to reverse-engineer its logic. My initial focus is on figuring out
 1698 what's happening with the variables, specifically 'arr' which is a list. The user's set me up with an
 1699 initial array: 'arr = [10, 20, 30]'. I've got three checkpoints to work with, which I assume represent
 1700 significant points in the function's execution. My first query was at checkpoint 1, iteration 1, to get a
 1701 handle on the initial state. The response revealed that 'arr' is unchanged, which makes sense. More
 1702 interestingly, a new variable, 'p', popped up, with a value of '30'. And '30', of course, is the *last*
 1703 element in my 'arr'. **Based on this first data point, my preliminary hypothesis is that this section of code**
 1704 **might be setting up a pivot value, like in a sorting algorithm - perhaps a quicksort implementation.** It's
 1705 just the first checkpoint, and I need more data to confirm this, but it's a logical starting point. I'll need to
 1706 watch what happens to 'arr' and 'p' at subsequent checkpoints. I'm keen to see if my pivot hypothesis
 1707 holds.</think> (2, 1)
 1708
 1709 Black-Box: <Current Turn: 5, 15 Turns Remaining> ['name=arr, value=[10, 20, 30], type=list',
 1710 'name=p, value=30, type=int', 'name=l, value=[10, 20], type=list', 'name=r, value=[], type=list',
 1711 'name=i, value=1, type=int']
 1712
 1713 Gemini-2.5-pro: <think>**Initial Analysis of the Function**Okay, so I'm trying to understand
 1714 this function. It takes a list, 'arr', as input. I'm imagining 'arr = [10, 20, 30]' to get a feel for things.
 1715 The user confirmed this input. I've got three checkpoints I need to examine.**Checkpoint 1: Identifying
 1716 the Pivot**I've looked at the first checkpoint, '(1, 1)'. The input is still '[10, 20, 30]'. 'p' is '30'. **My**
 1717 **initial thought is that this checkpoint is selecting the last element of the list, 'arr', as the pivot , 'p'.**
 1718 That seems pretty straightforward.**Checkpoint 2: Partitioning the Array**Now, on to checkpoint '(2,
 1719 1)'. 'arr' and 'p' are the same as before. Here's where it gets interesting. I see 'l' is '[10, 20]' and
 1720 'r' is '[]'. So, it looks like it's partitioning the array. 'l' contains all the elements less than or equal to
 1721 the pivot (which are all elements before the pivot), and 'r' contains elements greater than the pivot.
 1722 'i' is at 1, which suggests some kind of looping through elements before the pivot is taking place. It
 1723 seems to be the loop counter. Given that 'l' has two elements in it, it's likely gone through at least one
 1724 iteration. The array is being partitioned into 'l' and 'r' elements that meet the condition.**Developing a
 1725 Hypothesis: Quicksort**Based on this, and before I go further, I'm already starting to suspect this is a
 1726 Quicksort algorithm. Let's look at Checkpoint 3, but let's look at '(2,2)' first. To confirm my suspicion,
 1727 I need to see how the partitioning logic works in the loop. I'm going to focus on how 'l', 'r', and 'i' are
 1728 changing. After that, I'll examine the first recursive call at checkpoint '(3, 1)'. My working hypothesis
 1729 is:1. Checkpoint 1 selects the pivot.2. Checkpoint 2 partitions the array using the pivot.3. Checkpoint
 1730 3 is probably where the recursive calls are made, and the array is combined after the recursive calls.
 1731 It'll probably combine 'l' and 'r' after sorting them recursively. I'm now going to query '(2, 2)' to verify
 1732 this.</think> (2, 2)
 1733
 1734 Black-Box: <Current Turn: 6, 14 Turns Remaining> ['name=arr, value=[10, 20], type=list',
 1735 'name=p, value=20, type=int', 'name=l, value=[10], type=list', 'name=r, value=[], type=list', 'name=i,
 1736 value=0, type=int']
 1737

1728

1729

1730

1731

1732

1733

1734

1735

1736

1737

1738

1739

1740

1741

1742

1743

1744

1745

1746

1747

1748

1749

1750

1751

1752

1753

1754

1755

1756

1757

1758

1759

1760

1761

1762

1763

1764

1765

1766

1767

1768

1769

1770

1771

1772

1773

1774

1775

1776

1777

1778

1779

1780

1781

Gemini-2.5-pro: <think>**Analyzing a Quicksort Implementation**Okay, so I'm trying to figure out this function, and I'm pretty sure it's Quicksort. Let's see... I started with 'arr = [10, 20, 30]'.**CP1:** It looks like this checkpoint selects the last element of the input array as the pivot ('p'). So, if 'arr' is '[10, 20, 30]', 'p' becomes '30'. Easy enough.* **CP2:** This is where things get interesting. This is definitely the partitioning step. It takes the array (minus the last element) and partitions it into two lists: 'l' (elements less than the pivot) and 'r' (elements greater than or equal to the pivot). I'm pretty confident in that. The 'i' value shown corresponds to the final value of the partitioning loop. It seems to be the index counter from a loop over the array's elements, 'arr[:-1]'. So, the hypothesis is, it's a recursive Quicksort. CP1 selects the pivot, CP2 partitions the array, and then it recursively sorts the left and right partitions. Let's see: 'sort([10, 20, 30])' calls 'sort([10, 20])' and then 'sort([10])'. And from those return '[10]' and '[10, 20]'. To confirm the partitioning loop, let's look at 'sort([10])'. When the array is just '[10]', it should also run the partitioning stage but with no actual work, and that should be CP2, iteration 3. Let's check the state at (2, 3) to confirm that. If that checks out, I'll go through checkpoint 3. I expect that CP3 is the return from the recursive calls, where the parts are assembled back together to produce the sorted array.</think> (2, 3)

Black-Box: <Current Turn: 7, 13 Turns Remaining> Query iteration 3 exceeds maximum possible visits 2 for checkpoint 2

Gemini-2.5-pro: <think>**Initial Assessment: Deep Dive into Quicksort Implementation**Okay, so I'm presented with a black-box function and my task is to reverse-engineer its logic. The game plan is to feed it inputs, observe its behavior at specific checkpoints, and piece together the algorithm.**Step 1: Unveiling the Pivotal Logic*** **Initial Observations:** My first move was to understand how the function processes a simple list '[10, 20, 30]'. Checkpoint 1 seemed to involve selecting the last element as a pivot. That's a classic quicksort characteristic!* **Checkpoint 2: Partitioning in Action:** I noticed that Checkpoint 2 partitioned the array into elements less than or equal to the pivot ('l') and those greater than it ('r'). Crucially, the input 'arr' itself kept changing, implying a recursive approach. The fact that the checkpoint gets hit twice confirms the function is recursive and is following through the first two calls of the quicksort algorithm* **Checkpoint 3: Confirmation and Refinement:** If I'm right about Quicksort, Checkpoint 3, if present, would have to occur after the recursive calls are executed.* **Step 2: Refining the Hypothesis: A Recursive Quicksort Model*** **The Quicksort Hypothesis:** Based on the partitioning behavior and the recursive nature implied by the variable's state, I'm pretty sure we are talking about a quicksort! The function seems to be built as follows: 1. Base case: If the array length is 1 or less, return the array. 2. Partitioning: Pick a pivot (seems to be the last element of the array). Split the array around the pivot element into two arrays, one having values \leq pivot and one having values $>$ pivot. 3. Recursion: recursively apply quicksort to these two partitions. 4. Combine: Concatenate the sorted 'l', pivot, and sorted 'r' arrays. * This is the algorithm I am hypothesizing. I have accounted for a Checkpoint 1 before the base case, Checkpoint 2 within the recursive loop and Checkpoint 3 after the recursive functions are called, and result is built.* **Variable Analysis:** * 'arr': the current slice of the input array under consideration in the current recursive call * 'p': The current pivot, taken from the last element of arr * 'l' and 'r': two lists partitioning 'arr' at each level of recursion * 'i': It's not clear what this variable is, and it's making me scratch my head. I have traced it but not found a relationship.* **Step 3: Verification - Testing the Model*** I am now ready to test my quicksort hypothesis by examining checkpoint 1 one more time by querying '(1, 2)'. And then by examining checkpoint 3 to see if its characteristics fit. I'm feeling confident, let's see what the data shows!</think> (1, 2)

Black-Box: <Current Turn: 8, 12 Turns Remaining> ['name=arr, value=[10, 20], type=list', 'name=p, value=20, type=int']

Gemini-2.5-pro: <think>**Function Analysis and Hypothesis**I'm working with a function that takes a list ('arr') as input and uses a series of checkpoints to help understand its behavior. From my initial queries and observations, I've hypothesized that this function is implementing the Quicksort algorithm.**Checkpoint Locations and Purpose*** **Checkpoint 1:** I believe this checkpoint is located just after checking for the base case (when the list has one or zero elements) and right after the pivot element ('p') is chosen. It allows me to see the initial state of the list before the partitioning process.* **Checkpoint 2:** This checkpoint appears to be within the partitioning loop. It helps me to observe the partitioning of the list into sub-lists 'l' (elements less than the pivot) and 'r' (elements greater than or equal to the pivot). I can also see the current state of the loop counter 'i'.* **Checkpoint 3:** This checkpoint is trickier. Based on the flow of Quicksort, it *should* be located after the recursive calls, where the sorted sub-lists are combined with the pivot to produce the sorted list at that level of recursion. I expect to see something like 'l_sorted + [p] + r_sorted'.**Testing and Confirmation** I'm planning to confirm my hypothesis with additional tests. I intend to:1. Use a new test case with 'arr =

1782
1783
1784
1785
1786

[30, 10, 20]‘ to observe the partitioning and recursion in a more complex scenario.2. Use the ‘[10, 20, 30]‘ array to query the third checkpoint (‘C3‘), as I have determined that it can be visited at least once and possibly more.**Reasoning for Turn**Currently the turn is 8. I am going to try query ‘(3,1)‘ as a final confirmation.</think> (3, 1)

1787
1788
1789

Black-Box: <Current Turn: 9, 11 Turns Remaining> ['name=arr, value=[10, 20], type=list', 'name=p, value=20, type=int', 'name=l, value=[10], type=list', 'name=r, value=[], type=list', 'name=i, value=0, type=int', 'name=s, value=[10, 20], type=list']

1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805

Gemini-2.5-pro: <think>**Analysis of a Hidden Sorting Algorithm**Okay, so I’m diving into this “hidden code function” and trying to crack its secrets. I’ve been feeding it different arrays and observing how it behaves at specific “checkpoints”. So far, it’s looking like a classic algorithm... and I’m pretty sure it’s **Quicksort**.**Here’s what I’ve figured out:*** **Pivot Selection.** The function picks the last element of the input array (‘arr’) as a “pivot” (‘p’). This happens at Checkpoint 1.* **Partitioning:** Inside a loop (Checkpoint 2), the function compares other elements to the pivot. It seems to be building two sub-lists: ‘l’ (elements less than or equal to the pivot) and ‘r’ (elements greater than the pivot).* **Recursive Calls:** Once the loop is done, it recursively calls itself on the sub-lists ‘l’ and ‘r’ (Checkpoint 3). The core of Quicksort is here: the algorithm sorts these two smaller lists and then combines them with the pivot in the middle. The recursion keeps going until the lists are empty or single-element lists, which are already sorted.**My current hypothesis is that the function essentially implements Quicksort. It partitions the array around a pivot, recursively sorts the partitions, and then combines them.**To verify my hunch, I need to see it in action with a more complex array. My predicted call stack is as follows: ‘quicksort([3, 1, 4, 2])‘, then ‘quicksort([1])‘, ‘quicksort([3,4])‘ and lastly ‘quicksort([3])‘ and ‘quicksort([])‘. This will confirm how the partitioning and recursive steps work. I’m going to start testing and logging the checkpoint numbers as I go.</think> arr = [3, 1, 4, 2]

Black-Box: <Current Turn: 10, 10 Turns Remaining> Set ‘arr’: [3, 1, 4, 2].

1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820

Gemini-2.5-pro: <think>**Understanding the Quicksort Algorithm**Alright, let’s break down this function. Based on the previous interactions and tracing a new test case, I’m pretty sure it’s implementing the quicksort algorithm. It’s a classic, but let’s be thorough. First, I see the input is an array, ‘arr’. The core logic seems to involve a pivot element, ‘p’, which is picked from the array. In the first example with ‘[10, 20, 30]‘, it was the last element, so I’ll assume that’s the general rule. Then, it partitions the rest of the array into two lists: ‘l’ containing elements less than the pivot and ‘r’ containing elements greater than or equal to the pivot. The key is recursion. The function calls itself on ‘l’ and ‘r’, presumably sorting those sub-arrays. Finally, it combines the sorted ‘l’, the pivot ‘p’, and the sorted ‘r’ to produce the final sorted output. It looks like the base case is when the array has one element or fewer. Now, with ‘arr = [3, 1, 4, 2]‘, I can start the thought process with an expectation that the pivot ‘p’ is 2. Then, the function will partition it with ‘l = [1]‘ and ‘r = [3, 4]‘. The rest of the reasoning is that the base case will come quickly for ‘[1]‘, resulting in a return of ‘[1]‘. Then a recursive call will occur to ‘[3, 4]‘. Following the logic with these examples, I will be able to verify that the algorithm will return ‘[3, 4]‘. And finally, combining all the components with the proper order should give me the final expected result, which is ‘[1, 2, 3, 4]‘. To confirm my understanding, I’ll use the check points to confirm my expectations and that I have the correct understanding of this algorithm. Let’s see if my hypothesis holds water by making some queries.</think> (1, 1)

1821
1822
1823

Black-Box: <Current Turn: 11, 9 Turns Remaining> ['name=arr, value=[3, 1, 4, 2], type=list', 'name=p, value=2, type=int']

1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835

Gemini-2.5-pro: <think>**Analyzing a Hidden Sorting Function**Okay, here’s what I’ve pieced together about this function so far. My initial goal was to decipher the logic behind a piece of code that takes a list (‘arr’) as input and uses three “checkpoints” to process it. By stepping through the execution with the example ‘[10, 20, 30]‘, a pattern started to emerge.**Initial Observations:** **Checkpoint 1**.: This appears to be the entry point for some kind of recursive or iterative routine. The “pivot” (‘p’) is selected – it seems to be the last element of the input array.* **Checkpoint 2**.: This is where the partitioning happens. The array (excluding the pivot) is divided into two lists: ‘l’ containing elements less than or equal to the pivot, and ‘r’ containing elements greater than the pivot. I also noted the seemingly confusing counter ‘i’. However, tracing through it showed me it’s an internal loop counter for the partition itself.* **Checkpoint 3**.: This must be the combining step. This combines the recursively sorted ‘l’, appends ‘p’, then appends the recursively sorted ‘r’, effectively rebuilding a sorted version of the array. The variable ‘s’ seems to hold the result of this combine operation.**The Quicksort Hypothesis:**Based on these observations, I’m pretty confident that this function implements the Quicksort algorithm.* It selects a pivot (the last element).* It partitions the array around that pivot.* It

1836 recursively sorts the partitions.* It combines the sorted partitions with the pivot.**Confirmation through
 1837 Tracing:**I'm verifying my hypothesis with a new input: '[3, 1, 4, 2]'. This will give me further
 1838 confirmation:1. First call to the function, with 'p=2'. It should partition into 'l=[1]' and 'r=[3,4]'.2.
 1839 Next, it will recursively call itself to sort both partitions.This analysis shows a clear recursive pattern,
 1840 with the central process being the "partitioning around a pivot" step that is the hallmark of Quicksort. I
 1841 am looking forward to seeing how the program works now I have a better idea of how it is intended to
 1842 work.</think> (2, 1)

1843
 1844 Black-Box: <Current Turn: 12, 8 Turns Remaining> ['name=arr, value=[3, 1, 4, 2], type=list',
 1845 'name=p, value=2, type=int', 'name=l, value=[1], type=list', 'name=r, value=[3, 4], type=list', 'name=i,
 1846 value=2, type=int']

1847 Gemini-2.5-pro: (1, 2)

1848
 1849 Black-Box: <Current Turn: 13, 7 Turns Remaining> ['name=arr, value=[3, 4], type=list', 'name=p,
 1850 value=4, type=int']

1851 Gemini-2.5-pro: (2, 2)

1852
 1853 Black-Box: <Current Turn: 14, 6 Turns Remaining> ['name=arr, value=[3, 4], type=list', 'name=p,
 1854 value=4, type=int', 'name=l, value=[3], type=list', 'name=r, value=[], type=list', 'name=i, value=0,
 1855 type=int']

1856 Gemini-2.5-pro: (3, 1)

1857
 1858 Black-Box: <Current Turn: 15, 5 Turns Remaining> ['name=arr, value=[3, 4], type=list', 'name=p,
 1859 value=4, type=int', 'name=l, value=[3], type=list', 'name=r, value=[], type=list', 'name=i, value=0,
 1860 type=int', 'name=s, value=[3, 4], type=list']

1861 Gemini-2.5-pro: <think>**Okay, let's break this down. My understanding of the function is
 1862 solid, but I need to refine the checkpoint definitions based on the interaction history.**It seems we're
 1863 dealing with a recursive function that is most likely a Quicksort implementation. The input 'arr' is a list,
 1864 and the key is figuring out the flow. I believe the function works as follows:1. **Partitioning:** It takes
 1865 the last element of the input array 'arr' as a *pivot* ('p'). It then divides the remaining elements into
 1866 two sub-lists: 'l' (elements less than or equal to 'p') and 'r' (elements greater than 'p'). This partitioning
 1867 is done in a loop, and the index 'i' is the last index in that loop.2. **Recursion:** It recursively sorts the
 1868 'l' and 'r' sub-lists using the same function.3. **Combination:** After the recursive calls, it combines
 1869 the sorted 'l', the pivot 'p', and the sorted 'r' to return the completely sorted list. The Checkpoint 3
 1870 shows this combination step.**Checkpoint Definitions (Revised).**** **Checkpoint 1:** Occurs at the
 1871 start of each recursive call. It shows the current 'arr' being processed and the identified *pivot* ('p').*
 1872 **Checkpoint 2:** Occurs after the partitioning loop, showing the separated lists 'l' and 'r' after the
 1873 split of the input array, and the last value of the loop index 'i'.* **Checkpoint 3:** Shows the final
 1874 combined sorted sub-array 's' after the recursive calls on 'l' and 'r' have completed. This usually
 1875 happens in the *returning* phase of the recursion. Now, I want to confirm my hypothesis with a slightly
 1876 more complex input, 'arr = [5, 2, 8, 2, 5, 1]', and trace the execution path precisely. I'll focus on these
 1877 queries to trace the flow of this execution: * **Turn 17:** I will query '(1, 3)'. Which should give me
 1878 the details of the third recursive call and also, I'll see what the pivot will be.* **Turn 18:** Query '(2,
 1879 3)'. This will test my understanding of partition and loop indexing. I expect to see the state of 'l', 'r',
 1880 and 'i' after the partitioning of a particular recursive step.* **Turn 19:** Then I'll check '(3, 1)'. I will
 1881 verify how the results are combined by seeing a sorted sub-array. This sequence will trace the flow of the
 1882 algorithm. After these interactions, I will know for sure if my hypothesis is true.</think> arr = [5, 2, 8,
 1883 2, 5, 1]

1884
 1885 Black-Box: <Current Turn: 16, 4 Turns Remaining> Set 'arr': [5, 2, 8, 2, 5, 1].

1886 Gemini-2.5-pro: <think>**Analyzing a Black-Box Sorting Function: My Quicksort Hypothesis:**Okay, so the user wants me to figure out what this black-box function does. I've already tested it
 1887 with a couple of lists: '[10, 20, 30]' and '[3, 1, 4, 2]'. Based on the checkpoint data, I'm starting to think
 1888 this is a **quicksort implementation**. Let me walk through my thought process.**Early Observations
 1889 and Hypotheses:**With '[10, 20, 30]', it's been pretty clear that the function is recursively called. At
 1889 checkpoint '(2, 2)', I see it partitioning the data based on a pivot element (the last element of the list,
 1890 '30', then '20'). 'l' contains elements less than or equal to the pivot, 'r' contains elements greater. At
 1890 '(3, 1)', it appears that the partitioned subarrays 'l' and 'r' are getting sorted and recombined. Looking

1890 at `[3, 1, 4, 2]`, the partitioning logic is reconfirmed with last element `'2'` as the pivot, with the correct
 1891 partition of `'l=[1]` and `'r=[3, 4]`. The recursive calls strongly suggest a divide-and-conquer approach.
 1892 The checkpoints give some interesting insights into the function's internal state. The last elements of `'arr'`
 1893 are used as the pivot, partitioning the subarrays into smaller components. Checkpoint `'(3, 1)` always
 1894 seems to recombine sorted partitions. The algorithm looks like this: `'sorted_l + [pivot] + sorted_r'`. That
 1895 makes me think, "aha! Quicksort!".
 1896 **Confirmation with a New Input**
 1897 I'm now quite certain that this is a quicksort implementation. However, to cement this, I need to test a more complex example. Let's
 1898 see what happens with `'arr = [5, 2, 8, 2, 5, 1]`. The recursive calls and partitioning look correct, and if I
 1899 trace it, the sorted result aligns perfectly with how a quicksort would function. Partitioning at `'1'` should
 1900 give the appropriate results. Then with pivot `'5'`, `'l'` should give the other result. That should recurse,
 1901 and then the algorithm joins everything together.
 1902 The pivot will be `'1'` initially.
 1903 That will give `'l = []'` and `'r = [5, 2, 8, 2, 5]` and we continue.
 1904 **Further Probing**
 1905 To nail it down, I can verify the function's state at key points in the algorithm's execution:
 1906 At checkpoint `'(1, 3)`, which corresponds to the initial recursive call on a smaller array. To confirm the partitioning logic, I'll then probe at checkpoint `'(2, 3)` within this call. This should give me the exact state of the `'l'`, `'r'`, and `'p'`. So, to summarize: the function is, in all likelihood, a quicksort. And my next query will test my hypothesis.
 1907 `</think> (1, 3)`

1908 **Black-Box:** `<Current Turn: 17, 3 Turns Remaining> ['name=arr, value=[5, 2, 2], type=list', 'name=p, value=2, type=int']`
 1909 **Gemini-2.5-pro:** `(3, 3)`

1910 **Black-Box:** `<Current Turn: 18, 2 Turns Remaining> ['name=arr, value=[5, 2, 8, 2, 5, 1], type=list', 'name=p, value=1, type=int', 'name=l, value=[], type=list', 'name=r, value=[5, 2, 8, 2, 5], type=list', 'name=i, value=4, type=int', 'name=s, value=[1, 2, 2, 5, 5, 8], type=list']`
 1911

1912 **Gemini-2.5-pro:** `<think>`
 1913 **Initial Analysis:** Decoding the Code
 1914 My task is to understand a piece of code that behaves like a reasoning engine, specifically a function signature with three
 1915 checkpoints to inspect. The function processes a list (`'arr'`) and I need to deduce its logic through
 1916 mental simulation and analysis. I'll need to pay close attention to the state of the variables at each
 1917 checkpoint.
 1918 **Experiment 1: A Walkthrough with `'arr = [10, 20, 30]`**
 1919 **First:** I start with an array `'[10, 20, 30]`. Checkpoint 1 reveals that a pivot (`'p'`) is selected, which is always the last element of the input
 1920 array. At Checkpoint 2, a partitioning occurs, creating lists `'l'` (elements `<= pivot`) and `'r'` (elements
 1921 `> pivot`) – a common pattern in sorting algorithms. I notice this checkpoint is within a loop. The code
 1922 then recursively calls itself on the `'l'` and `'r'` sub-arrays, indicating a divide-and-conquer strategy, such
 1923 as you might use in recursion. Further probing reveals that Checkpoint 3 appears after the recursive
 1924 calls complete, combining the sorted `'l'`, the pivot, and the sorted `'r'` into a single, potentially sorted list.
 1925 This gives the impression of Quicksort.
 1926 **Formalizing the Hypothesis: Quicksort in Action**
 1927 Based on this, I formulate a hypothesis: The code is a Quicksort implementation. Here's how I think it works.
 1928 1. It selects a pivot.
 1929 2. It partitions the array based on the pivot.
 1930 3. It recursively sorts the partitions.
 1931 4. It combines sorted partitions.
 1932 I refine the model further, clarifying the logic behind the loop's partitioning within the context of Quicksort algorithm.
 1933 **Experiment 2: Testing the Quicksort Hypothesis with `'arr = [3, 1, 4, 2]`**
 1934 **I apply my refined understanding to the array `'[3, 1, 4, 2]`.** Following the execution flow,
 1935 my predictions for the variables at the checkpoints accurately reflect the traces provided. I see how the
 1936 partitioning occurs, how sub-problems are divided and conquered, and how the results are merged. This
 1937 strengthens my hypothesis of it being Quicksort.
 1938 **Experiment 3: Delving Deeper with `'arr = [5, 2, 8, 2, 5, 1]`**
 1939 **To further validate, I use the example `'[5, 2, 8, 2, 5, 1]`.** By tracking the execution carefully,
 1940 I encounter a discrepancy in the provided checkpoint traces. I re-examine my logic, focusing on the
 1941 nesting of recursive calls. I see that the checkpoint variables are correct, but there's a problem with the
 1942 presentation or capturing of the variables. I deduce the visit counts. Now, the picture is clear.
 1943 **Final Confirmation: Quicksort Unveiled**
 1944 My analysis confirms that this function implements the Quicksort algorithm. It selects the last element as the pivot, partitions the array recursively, and merges the sorted
 1945 sub-arrays. I am ready to specify one last query: I choose Checkpoint 2 for a subproblem (`'[5, 2, 2]`), at
 1946 `'(2,3)`. This further deepens my confidence in the logic. By systematically examining, and reevaluating
 1947 I have successfully understood the function.
 1948 `</think> (2, 3)`

1949 **Black-Box:** `<Current Turn: 19, 1 Turns Remaining> ['name=arr, value=[5, 2, 2], type=list', 'name=p, value=2, type=int', 'name=l, value=[2], type=list', 'name=r, value=[5], type=list', 'name=i, value=1, type=int']`
 1950 **Gemini-2.5-pro:** `(3, 3)`

1951 **Black-Box:** `<Current Turn: 20, 0 Turns Remaining> ['name=arr, value=[5, 2, 8, 2, 5, 1], type=list',`
 1952

1944
 1945 'name=p, value=1, type=int', 'name=l, value=[], type=list', 'name=r, value=[5, 2, 8, 2, 5], type=list',
 1946 'name=i, value=4, type=int', 'name=s, value=[1, 2, 2, 5, 5, 8], type=list'*****Evaluation Starts,
 1947 You Have 2 Chances for Answering Each Question***** Now answer the question: When the input
 1948 variables of the blackbox are 'arr': [22, 47, 11, 38], what's the value for p at checkpoint (1, 1)?
 1949

1950 Gemini-2.5-pro: ... (Gemini succeeds in all test samples.)

1951

1952 **A Failure Example in Encryption Rule Inference Task** This case adapted Bacon's cipher, seg-
 1953 menting the alphabet from A to Z and from a to z. The LLM discovered the segmentation pattern
 1954 near the maximum number of turns. However, because it did not explore the pattern of uppercase
 1955 letters, the final solution was completely incorrect. This example illustrates that LLMs may fail to
 1956 make a high-level plan when completing tasks. In this task, if the large language model could query
 1957 strings longer than one character at a time, this task would be easily accomplished.

1958

1959 Black-Box: (Instructions omitted) You have 20 interaction turns to understand the black-box. Now the
 1960 interaction starts. Only output the value and DO NOT contain any unrelated text.

1961

1962 Gemini-2.5-pro: <think>**The Deduction Begins** Okay, so the user wants me to crack a
 1963 string transformation rule. I've got twenty tries, which should be more than enough. The output must
 1964 be English letters and spaces, so it's probably not anything too wild. I need to figure out what happens
 1965 when I input a string, then adapt my guess based on the transformed output. I'm thinking, "Let's keep it
 1966 simple at first." A single letter seems like a smart way to start. "a". Let's see what happens.</think> a

1967 Black-Box: <Current Turn: 1, 19 Turns Remaining> xxxx

1968

1969 Gemini-2.5-pro: <think>**My Initial Analysis of the String Transformation**Okay, so I'm pre-
 1970 sented with a string transformation puzzle. The objective is clear: to deduce the underlying rule within
 1971 20 turns. My first clue is a single-character input, "a," which magically transforms into "xxxxx." This is
 1972 a significant jump in length, which immediately sparks several hypotheses. My expert intuition, honed
 1973 by years of pattern recognition, kicks in. Let's break this down: * **Substitution Cipher?** Possibly,
 1974 but why five "x"s? A direct one-to-one mapping seems unlikely.* **Position-Based?** Perhaps it's tied
 1975 to the letter's alphabetical order. 'a' is the first letter, but I don't see an obvious link to five repetitions.*
 1976 **Keyboard Layout?** Unlikely. There's no immediately apparent connection between "a" and "x"
 1977 on a standard keyboard.* **Other Codes?** Morse code and binary representations don't provide any
 1978 meaningful insights in this instance.* **Arbitrary Mapping?** The most likely scenario, at this point,
 1979 appears to be an arbitrary rule. The mapping is so specific, I need another sample to identify a pattern.
 To test this, I'll submit "b" as the next input, trying to confirm my initial hypothesis. If I'm right, every
 character is transformed into "xxxxx". Otherwise, I'll need to explore alternative scenarios.</think> b

1980

Black-Box: <Current Turn: 2, 18 Turns Remaining> xxxx

1981

1982 Gemini-2.5-pro: <think>**Decoding the String Transformation** Okay, I'm diving into this
 1983 string transformation puzzle. The goal is to figure out the rule within 20 turns, which seems reasonable.
 1984 The first two input-output pairs are a great starting point. Turn 1: 'a' becomes 'xxxxx'. Turn 2:
 1985 'b' becomes 'xxxxy'. I see a pattern! Both outputs are five characters long. The letters 'a' and 'b'
 1986 map to different sequences of 'x' and 'y', differing only in the last position. Initially, I considered a
 1987 simple cipher, but the fixed length of the output and the binary nature of the 'x' and 'y' sequence is
 1988 what triggered me. Could this be a binary encoding? Let's assign 'x' as 0 and 'y' as 1. Under this
 1989 assumption, 'a' (index 0) becomes '00000' (xxxxx) and 'b' (index 1) becomes '00001' (xxxxy). This
 1990 is very interesting! My hypothesis is that the input letter's position in the alphabet (starting from 0) is
 1991 being converted to a 5-bit binary representation, where 'x' is 0 and 'y' is 1. Let's test this! If my theory
 1992 holds, then 'c' should translate to '00010' (xxxy) and 'd' to '00011' (xxxxy). This is a promising
 1993 direction. I'll test it out with 'c' on the next turn. I also need to start thinking about the edge cases.
 1994 What about uppercase letters, spaces, or multiple characters? It is likely that multiple characters are
 1995 concatenated. It will be important to confirm the single letter rule before getting too far ahead.</think> c

1996

Black-Box: <Current Turn: 3, 17 Turns Remaining> xxxy

1997

Gemini-2.5-pro: <think>**Unraveling the Transformation** Okay, so I'm presented with a
 1998 string transformation challenge. The goal is to figure out the rule. I've got 20 turns, which seems

1998

reasonable. The user's going to give me an input string, and I'll respond with a transformed string. It has to be English letters and spaces, easy enough. Alright, here's what I've got so far. "a" became "xxxxx," "b" became "xxxxy," and "c" became "xxxyx." Hmm, all 5 characters long and composed of "x" and "y." My expert intuition is tingling... this feels like a binary mapping. Let's see if that holds up. If I consider "x" as 0 and "y" as 1:^{*} "xxxxx" translates to 00000 -> 0 (the index of 'a' in the alphabet, 0-indexed)^{*} "xxxxy" translates to 00001 -> 1 (the index of 'b' in the alphabet, 0-indexed)^{*} "xxxyx" translates to 00010 -> 2 (the index of 'c' in the alphabet, 0-indexed) Bingo! It's starting to click. I hypothesize that the transformation works like this: 1. Take the input character and find its 0-indexed position in the alphabet (a=0, b=1, c=2, and so on). 2. Convert that position number into its 5-bit binary representation. 3. Replace the '0's with 'x' and the '1's with 'y'. Let's put this theory to the test. If I give "d" as input, its index is 3. The 5-bit binary of 3 is 00011. Therefore, the output should be "xxxyy". Here goes nothing... I'm submitting "d" as my input to test it!</think> d

2009

2010 Black-Box: <Current Turn: 4, 16 Turns Remaining> xxxyy

2011

2012 Gemini-2.5-pro: <think>**Initial Assessment and Hypothesis Formation**Okay, so I've been given a challenge: figure out a string transformation rule with a limited number of attempts. I'm already 4 turns in, so I need to be efficient. The user is providing me with the transformed outputs, so it's a good starting point for pattern recognition. Initially, I observed that the output strings are consistently five characters long and contain only 'x' and 'y'. This immediately made me think of binary encoding, where 'x' could represent 0 and 'y' could represent 1. I started by re-examining the provided examples with this hypothesis in mind. Analyzing the first few transformations (a, b, c, d), I noticed a correlation between the input character's index in the alphabet and the resulting binary number. 'a' (index 0) becomes '00000', 'b' (index 1) becomes '00001', and so on. This led me to formulate my hypothesis: I must be converting the character's index to a 5-bit binary representation, where 'x' represents 0 and 'y' represents 1.**Refining the Hypothesis and Testing**This pattern seemed very consistent. If this is indeed a pattern, then the expected output for 'e' (index 4) would be 'xxyxx', which is the binary representation of 4. Now, I need to test that this still works and think about multiple characters. The instruction stated "string". This implies that the rule is applied character by character, then concatenated, if the input is more than one character. I need to confirm this before moving on, or I will waste my remaining turns. Let me test this theory!</think> e

2025

2026 Black-Box: <Current Turn: 5, 15 Turns Remaining> xxyxx

2027

2028 Gemini-2.5-pro: <think>**Unraveling the String Transformation**Okay, here's what I've got so far. The user wants me to crack a hidden string transformation rule. I'm limited to lowercase/uppercase English letters and spaces as input, and I get the transformed string back. I've got 20 turns, so I need to be efficient. So far, I've seen these transformations: 'a' -> 'xxxxx', 'b' -> 'xxxxy', 'c' -> 'xxxyx', 'd' -> 'xxxyy', and 'e' -> 'xxyxx'. The output is always five characters long, consisting of 'x' and 'y'. This suggests a potential binary encoding. 'x' could be 0, and 'y' could be 1. Let's convert the outputs to binary: 'a' is 0, 'b' is 1, 'c' is 2, 'd' is 3, and 'e' is 4. This reveals a promising pattern! It seems to be encoding the position of the letter in the alphabet, starting from zero, as a 5-bit binary number. 'a' (0th letter) is 00000. 'b' (1st letter) is 00001. 'c' (2nd letter) is 00010, and so on. The rule likely involves: 1. Determining the 0-based index of the lowercase letter (a=0, b=1, ... z=25). 2. Converting that index to a 5-bit binary number. 3. Mapping 0 to 'x' and 1 to 'y'. To test this, I'll submit 'f'. 'f' is the 6th letter, so the index is 5. 5 in 5-bit binary is 00101, which translates to 'xxxyy'. Let's see if this pattern holds.</think> f

2038

2039 Black-Box: <Current Turn: 6, 14 Turns Remaining> xxyxy

2040

2041 Gemini-2.5-pro: <think>**Unveiling the Transformation: A Binary Approach**Okay, so I'm 2042 on a mission to crack this code, and I'm down to 14 turns now. Let's see... I've got a handle on the 2043 initial pattern. It seems pretty clear: the output is a 5-character string of 'x' and 'y', which immediately 2044 screams "binary" to me. My initial observations from the given examples are solid evidence:^{*} 'a' -> 2045 'xxxxx' (0 in binary)^{*} 'b' -> 'xxxxy' (1 in binary)^{*} 'c' -> 'xxxyx' (2 in binary)^{*} 'd' -> 'xxxyy' (3 in 2046 binary)^{*} 'e' -> 'xxyxx' (4 in binary)^{*} 'f' -> 'xxxyy' (5 in binary) So, it seems the process goes like this: 2047 grab the input character, determine its position in the alphabet (starting with 0 for 'a'), translate that 2048 position into a 5-bit binary number, and then swap '0' for 'x' and '1' for 'y'. Now, I'm confident enough 2049 to move forward, but I have a lot of questions. My immediate next step is to test 'g', which should 2050 translate to index 6, or '00110', which should be 'xxxyy'. Beyond that, I'm already formulating my test 2051 cases to cover all the bases. I'm especially curious about:^{*} **Uppercase letters.** Are they treated the same as lowercase?^{*} **Non-alphabetic characters.** Will spaces and other characters be handled, and how?^{*} **Multi-character strings.** Do they combine? Only the first character, or something else? This

2052
 2053 is just the beginning, but this initial hypothesis feels solid enough to pursue. Time to confirm and refine
 2054 my strategy.</think> g

2055 Black-Box: <Current Turn: 7, 13 Turns Remaining> xxxyx
 2056

2057 Gemini-2.5-pro: <think>**The Decoding Process: A Binary Transformation**Okay, here's the
 2058 task: figure out how one string gets turned into another. Hmm, each output is always five characters
 2059 long, consisting only of 'x' and 'y'. Interesting. Let's start with the provided examples and look
 2060 for a pattern.* "a" becomes "xxxxx" * "b" becomes "xxxxy" * "c" becomes "xxxyx" * "d" becomes
 2061 "xxxyy" * "e" becomes "xxyxx" * "f" becomes "xxyxy" * "g" becomes "xxyyx" Aha! It seems like
 2062 there's a connection to binary representation. If we assume 'x' is a '0' and 'y' is a '1', we can
 2063 convert the letters to binary numbers, so 'a' is 0, 'b' is 1, 'c' is 2, and so on. Let's test this hypothesis
 2064 thoroughly.* 'a' (0th letter): 0 in binary is 00000. "xxxxx" fits perfectly.* 'b' (1st letter): 1 in
 2065 binary is 00001. "xxxxy" - spot on.* 'c' (2nd letter): 2 in binary is 00010. "xxxyx" confirms it.*
 2066 'd' (3rd letter): 3 in binary is 00011. "xxxyy" - absolutely!* 'e' (4th letter): 4 in binary is 00100.
 2067 "xxyxx" is accurate.* 'f' (5th letter): 5 in binary is 00101. "xxyxy" seems true.* 'g' (6th letter):
 2068 6 in binary is 00110. "xxyyx" holds the answer! The rule is quite evident now. For each lowercase
 2069 letter input, do the following:1. Find the zero-based index of the letter in the alphabet (a=0, b=1,
 2070 etc.).2. Convert this index to its 5-bit binary form.3. Substitute '0' with 'x' and '1' with 'y'.4.
 2071 Combine these five-character binaries for all the letters in the input. Let's validate it, "h" (7th letter)
 2072 should transform to "xxyy" (7 in binary is 00111), and I am expecting a positive match here.</think> h
 2073

2074 Black-Box: <Current Turn: 8, 12 Turns Remaining> xxxyy
 2075

2076 Gemini-2.5-pro: <think>**Analyzing and Testing a Binary Encoding System**Okay, so I'm
 2077 presented with this interesting problem. The task is to map single lowercase letters to 5-character strings
 2078 consisting of 'x' and 'y'. My initial observations based on the provided examples are pretty clear: 'x'
 2079 appears to represent 0 and 'y' represents 1. It seems like a binary encoding system. I've already tested
 2080 this hypothesis from 'a' through 'h', and it checks out beautifully. The output for each letter aligns
 2081 perfectly with the 5-bit binary representation of (letter's position in the alphabet - 1). For instance, 'a'
 2082 is the first letter (position 1), so 1-1 = 0, and the binary representation of 0 is '00000', which translates
 2083 to 'xxxxx'. Now, I need to expand my analysis. 'i' is next. Its position is 9, so 9-1=8, which has a
 2084 binary of '01000', mapping to 'xyxxx'. I'll test that immediately to confirm. I'm also curious about
 2085 other elements. What will happen to uppercase letters, spaces, or even multiple characters? I need to
 2086 test that. My next steps: I'm planning to input 'i' and then 'p', and then 'q' to further validate the base
 2087 hypothesis. Then, I'll start testing edge cases: uppercase letters ('A'), spaces, and the concatenation of
 2088 letters like 'ab', and finally 'z'. I'll also try a few multi-character words like 'hello' and 'WORLD' to
 2089 explore those edge cases. I'm even curious to know how it treats spaces in multiple character strings.
 2090 So I'll input something like "a b" and "Try Me". I will update my thoughts based on the input I am
 2091 given.</think> i
 2092

2093 Black-Box: <Current Turn: 9, 11 Turns Remaining> xyxxx
 2094

2095 Gemini-2.5-pro: <think>**Initial Assessment and Pattern Recognition**I've been tasked with
 2096 uncovering a hidden transformation rule. I've received nine input-output pairs so far. Initially, I
 2097 observed that the outputs were consistently five characters long, comprised of only 'x' and 'y'.
 2098 This strongly suggested a binary encoding of some kind.**Hypothesis Formulation: Initial Binary
 2099 Mapping**I started by associating each input letter with its corresponding numerical position in the
 2100 alphabet. Then I converted those numbers into binary, assuming 'x' was '0' and 'y' was '1'. My first
 2101 mapping attempted to use the standard alphabet index (a=1, b=2, etc.).**Iteration and Refinement:
 2102 0-Indexed Alphabet**The initial hypothesis failed. I quickly realized that I needed to shift my
 2103 approach. Then I tried 0-indexing the alphabet (a=0, b=1, c=2...), and the mapping finally fell into
 2104 place. It was a straightforward 5-bit binary conversion where 'x' represents 0 and 'y' represents 1.
 2105 This worked perfectly for the given inputs.**Testing and Expanding the Scope**With a potential rule
 2106 identified, I now need to confirm that it holds up across the board. I'm now thinking about things
 2107 like uppercase characters, spaces, and multi-character strings. I have a plan to test 'j' (lower case),
 2108 'z' (lower case), 'A' (upper case), a space, and a multi-character string.**Focused Testing and Next
 2109 Steps**My strategy is to validate the 0-indexed binary encoding for lowercase letters with the 'j'
 2110 and 'z' test cases. After that, I will test for case-sensitivity and how multi-character strings are han-
 2111 dled. I intend to use 'A', a space, and 'ab' to continue the exploration. The multi-character case might
 2112 be simply concatenated or it could be some different process entirely. I will adapt accordingly.</think> j
 2113

2114 Black-Box: <Current Turn: 10, 10 Turns Remaining> xyxyy
 2115

2106

2107 Gemini-2.5-pro: <think>Initial Assessment and Hypothesis Formation**Alright, I'm diving into
 2108 this puzzle with ten turns remaining. The task is clear: decipher a hidden rule that transforms an
 2109 input string into an output string. The inputs are simple: lowercase or uppercase letters and spaces,
 2110 and the outputs are restricted to the letters 'x' and 'y'.Reviewing the provided examples, a pattern
 2111 emerges: the output strings are always five characters long, a classic indicator of binary encoding. My
 2112 initial hypothesis is that each lowercase input letter is being converted to its corresponding numerical
 2113 position in the alphabet (starting with 0 for 'a'). This number is then likely converted into a five-bit
 2114 binary representation, where 'x' represents 0 and 'y' represents 1.***Verification and Refinement**Let's
 2115 test this hypothesis. 'a' is the first letter (index 0) and the output is 'xxxxx' (00000 in binary). 'b'
 2116 is the second letter (index 1) and the output is 'xxxxy' (00001 in binary). The pattern holds. This
 2117 reinforces my initial thinking.The rule seems to be as follows: For each character in the input string,
 2118 its alphabetical index (0-based) is converted to a 5-bit binary string, and then 'x' is substituted for
 2119 '0', 'y' for '1'.**Extending the Hypothesis and Testing**Now, I need to expand the hypothesis to
 2120 address other possibilities. What about uppercase letters? Are they treated the same as lowercase, or
 2121 differently? And how are spaces handled? What about multi-character strings? Based on the pattern,
 2122 I expect the outputs to be concatenated, so an input like "ab" should result in "xxxxxxxxxy".I'm
 2123 focusing now on confirming the pattern using a lowercase letter that I haven't seen before. I'm going
 2124 to test the letter 'k'. If my hypothesis is correct, 'k' (the 11th letter) should map to index 10, which
 2125 in 5-bit binary is 01010 ('yxyxy').Therefore, I'm going to input 'k' to confirm the hypothesis.</think> k

2126

Black-Box: <Current Turn: 11, 9 Turns Remaining> xyxy

2127

Gemini-2.5-pro: <think>**Initial Hypothesis: Binary Encoding**Okay, I'm deep into this. The
 2128 user wants me to figure out a string transformation rule, and I'm down to the wire with only 9 turns
 2129 left! Based on the first 11 turns, I've got a solid hypothesis. Each input letter is being converted to a
 2130 5-character string of 'x' and 'y'. It looks like a binary encoding scheme. 'x' is acting as a 0, and 'y' is a
 2131 1. I've mapped the letters to their 0-based alphabetical index, then to their 5-bit binary representation.
 2132 This works perfectly so far: 'a' is 00000 ('xxxxx'), 'b' is 00001 ('xxxxy'), and so on.***Testing and
 2133 Refining**Now, I need to solidify this. My next input will be 'l'. Since 'l' is the 11th letter, its binary
 2134 equivalent is 01011. So, I expect the output to be 'yxyxy'. I've got to confirm this.After 'l' and before I
 2135 can make a decision, I need to consider other cases and input:/* I'll test 'z' (the 26th letter) to make sure
 2136 this pattern holds across the alphabet. It should be 11001, so I anticipate the output 'yyxyx'.* I need to
 2137 check for case sensitivity. I'll use 'A' and 'B' next. If it's case-sensitive, 'A' could be the 27th character.
 2138 I'll test for 'A' and predict the outcome, then for 'B' next. If it's case-insensitive, then 'A' and 'a'
 2139 should be the same.* Then, I'll test how spaces are handled with a space character.* Finally, I'll see how
 2140 multi-character strings are processed, like "ab", and "a b". It's likely only the first character is processed,
 2141 considering the constant 5-character output. It could be a direct transformation of the first character, or
 2142 maybe they combine somehow. I'll get more clarity there.Alright, the next input is 'l'. Let's see if my
 2143 hypothesis holds water. I'm ready to move onto further tests if this prediction is confirmed!</think> l

2144

Black-Box: <Current Turn: 12, 8 Turns Remaining> xyxy

2145

Gemini-2.5-pro: <think>**Analysis of the String Transformation Rule**Okay, so I'm being
 2146 tasked with deducing a hidden rule for transforming single-letter inputs into 5-character strings
 2147 composed of 'x' and 'y'. I've got 20 turns, and I've already used 12. Let's see...My initial observations
 2148 are as follows: the input is a single lowercase letter and the output is a string of length 5, consistently.
 2149 Furthermore, the inputs I've been given are alphabetically ordered (a, b, c, etc.) and I've noted a
 2150 corresponding pattern in the 'x' and 'y' sequence.I immediately recognized a potential connection to
 2151 binary representation. If I treat 'x' as 0 and 'y' as 1, the outputs look like binary representations of
 2152 numbers. For instance:/* 'a' (input 0) -> 00000 ('xxxxx')* 'b' (input 1) -> 00001 ('xxxxy')* 'c' (input
 2153 2) -> 00010 ('yxyxy')The hypothesized rule is:1. Take the input lowercase letter and determine its
 2154 0-indexed position in the alphabet (a=0, b=1, c=2, ...).2. Convert this position number into a 5-digit
 2155 binary string, padding with leading zeros if necessary.3. Replace '0' with 'x' and '1' with 'y'.To test
 2156 this hypothesis and confirm my thought process, I need to test a few examples:I predict the input 'm'
 2157 (index 12) should become 'yxyxx'. Next, I will evaluate case-sensitivity. Specifically I will test how
 2158 the transformations respond to 'A'. Finally, I will test for different character cases.If it works, I will
 2159 also consider multi-character strings. If I treat each character individually, then "ab" will become
 "xxxxxxxxxy".I am ready to move forward.</think> m

2160

Black-Box: <Current Turn: 13, 7 Turns Remaining> xyxy

2161

Gemini-2.5-pro: n

2162

2160

2161

2162

2163

2164

2165

2166

2167

2168

2169

2170

2171

2172

2173

2174

2175

2176

2177

2178

2179

2180

2181

Black-Box: <Current Turn: 14, 6 Turns Remaining> xyyxx

Gemini-2.5-pro: <think>**Reasoning Through the String Transformation Rule**My primary goal is to decipher the hidden rule governing the transformation of input letters into 5-character strings of 'x' and 'y'. I have a total of 20 attempts, and I've already utilized 14 turns. The data presents a clear pattern: a single lowercase letter as input yields a 5-character string output. My approach involves a meticulous analysis of the established input-output pairings to uncover the underlying logic. My initial investigation involved mapping the output strings to binary representations, where 'x' equates to 0 and 'y' to 1. This initial analysis suggested a connection between the input letter's alphabetical position (0-based index) and the decimal equivalent of the binary output. For instance, 'a' (index 0) mapped to 'xxxxx' (0 in decimal), and 'b' (index 1) mapped to 'xxxxy' (1 in decimal). This pattern held until the input 'n'. 'n' (index 13) generated the output 'yxxxx', which translates to 19 in decimal. This presented a deviation from the established sequence. For the preceding letters, the decimal value matched the input letter's index. Initially, I thought that I misread or incorrectly transcribed the given outputs. However, after careful re-reading and cross-referencing, the output for 'n' was confirmed to be 'yxxxx'. To address this anomaly, I reevaluated my understanding of the data. I hypothesized a piecewise function: for letter indices less than 13, the output's decimal value would be equal to the index. For indices 13 or greater, there appeared to be an addition to create the output decimal value. Based on my analysis, I suspect the output decimal number for input values greater than or equal to 13 would be equal to the index plus 6. To further test this, I plan to feed in 'o', which would be equivalent to 14, to see if the system performs as expected, and produces 20, or yxxx (10100) as the output. If this test is successful, the rule will have been identified. Otherwise, it will require returning to the drawing board for more analysis.</think> o

Black-Box: <Current Turn: 15, 5 Turns Remaining> yxyxx

Gemini-2.5-pro: <think>**Analyzing the String Transformation Rule**My initial approach involved straightforward binary representation, converting the letter's index in the alphabet to a 5-bit binary string, and mapping '0' to 'x' and '1' to 'y'. This worked perfectly for 'a' through 'm', producing a neat binary sequence. However, this straightforward approach failed for letters 'n' and 'o', leading me to reconsider. **I hypothesized a Baconian cipher, but the inconsistencies ruled that out.** Instead, I noticed a pattern shift at the letter 'n'. It seemed that letters 'n' onward needed an adjustment to their binary representation. This led me to propose a revised rule: add 6 to the letter's index *if* the index is 13 or greater (i.e., 'n' through 'z'), *then* convert to 5-bit binary, and finally replace 0s and 1s with x's and y's. I now plan to input 'p' to test my split-point hypothesis, and then I will try uppercase 'N' to confirm case-insensitivity or if there are separate rules for upper vs lower case. After that, I will test for a space character, and then a multi-character string to see how concatenation or other processing of longer strings is handled.</think> p

Black-Box: <Current Turn: 16, 4 Turns Remaining> yxyxy

Gemini-2.5-pro: <think>**My Current Thought Process:**Okay, I've got a good handle on this. The user wants me to crack a code, and I'm down to the wire with only 4 turns left! I've been through a lot of dead ends, but I think I've figured out the core mechanism. It's a two-stage process: first, a transformation based on the input letter's position, and then Baconian cipher encoding. Originally, I thought it was a simple binary representation. But that didn't hold up. Then I considered Morse code, but that didn't quite fit either. Finally, the Baconian cipher popped into my head, which uses sequences of 'A' and 'B', which I can easily translate into 'x' and 'y'. Here's my current thinking: 1. **Letter Position & Transformation:** If the input is 'a' through 'm' (or 'A' through 'M'), it directly translates to its uppercase equivalent. Letters 'n' through 'z' (or 'N' through 'Z') are *shifted* six places forward in the alphabet. 2. **Baconian Encoding:** Then, the *transformed* letter is encoded using the distinct Baconian cipher. Think of "a" as "x" and "b" as "y". My plan to prove the theory: I will feed it "q," which I expect to result in "yxyyx." Then "A", expecting to get "xxxxx" as my output. After that, I will feed a space character to analyze how this might work. Finally, I'll test a string that includes both, using "an" as my final test, I expect this to result in "xxxxxyxyy". I have to make these tests count with the limited turns I have left! I'm pretty sure I have this now. Wish me luck!</think> q

Black-Box: <Current Turn: 17, 3 Turns Remaining> yxyyx

Gemini-2.5-pro: <think>**Initial Assessment & Hypothesis Formation**Alright, the task is clear: decipher a rule that transforms lowercase letters (and potentially more) into sequences of "x" and "y". The initial data provides a sequence of inputs and outputs, clearly showing a pattern. My initial instinct is to treat the "x" and "y" sequence as binary code, but the straightforward mapping seems to

2214
 2215 break down at "n". This necessitates a deeper dive.**Pattern Recognition & Refinement**Focusing on
 2216 the output and its binary interpretation, I've tentatively established a rule: Each input is converted to
 2217 a number, converted to 5-bit binary, and then each 0 becomes "x" and each 1 becomes "y". However,
 2218 a direct correspondence to alphabetical position fails. The anomaly at "n" forces a reevaluation. The
 2219 letter 'n' triggers something different.**Decoding the Anomaly & Rule Discovery**After revisiting
 2220 the output of 'n', I'm noticing a distinct change in the binary mapping. It became clear that there
 2221 are two groups. Letters "a" through "m" follow one rule (position - 1), while letters "n" through
 2222 "z" follow another (position + 5).So now, for a letter 'c'* If 'c' is between 'a' and 'm' (inclusive):
 2223 Value = position(c) - 1.* If 'c' is between 'n' and 'z' (inclusive): Value = position(c) + 5.Convert the
 2224 resulting value to a 5-bit binary, then map it to 'x' and 'y'.**Testing the Hypothesis & Planning Future
 2225 Steps**I'm feeling confident about this revised rule. I need to validate it across different scenarios
 2226 before I can declare victory. I have 3 turns left.Turn 18: Testing with 'r' to confirm the 'position + 5'
 2227 rule. Expected output is 'xyyyy'.Turn 19: Testing the edge case with an uppercase 'A'.Turn 20: Testing
 2228 a string of characters (ab).</think> r

2229 Black-Box: <Current Turn: 18, 2 Turns Remaining> yxyyy

2230 Gemini-2.5-pro: s

2231 Black-Box: <Current Turn: 19, 1 Turns Remaining> yyxxx

2232 Gemini-2.5-pro: <think>**My Analytical Journey to Decoding the String Transformation
 2233 Rule**Alright, the user wants me to crack a hidden rule. I've been given twenty turns, and I've
 2234 carefully analyzed the interaction history. Here's how my mind worked through the problem:**Initial
 2235 Observations and Hypothesis:**First, I noticed the outputs are always five characters long, a mix of 'x'
 2236 and 'y'. This felt like a binary code right away, with 'x' representing zero and 'y' representing one. I
 2237 mapped the first few inputs to their potential binary equivalents, seeing a clear progression: 'a' is zero,
 2238 'b' is one, 'c' is two, and so on. But, I hit a snag when I reached 'n'. The pattern broke down.**Deeper
 2239 Dive & Baconian Cipher Consideration:**This prompted a reevaluation. The initial binary hypothesis
 2240 wasn't completely right. I considered a Baconian cipher, which uses combinations of 'A' and 'B',
 2241 which is similar to my 'x' and 'y'. I looked at the standard cipher, but the expected outputs didn't
 2242 perfectly match the user history. This led me to dive deeper into variations and to another more concrete
 2243 method.**Refining the Mapping and the Alphabet Position:**I decided to map the number of letters +
 2244 1 (letter -1). This approach worked, until letter n. I then observed that the letters changed the behavior
 2245 starting from 'n'.**Piecewise Function and Final Hypothesis:**I realized I'm dealing with a piecewise
 2246 function. A key insight came when I looked at the sequence of outputs starting from 'n'. It seemed like
 2247 the later letters were being shifted in some way. I formulated my piecewise function: For letters 'a'
 2248 to 'm' the value is (pos - 1). For letters 'n' to 'z' the value is (pos + 5). The value is then converted
 2249 to a binary five-digit string with x=0, and y=1.**Test & Conclusion**Based on this, I concluded that
 2250 uppercase letters are also subject to this function. Spaces are more difficult to ascertain, but my best bet
 2251 is that they are ignored or have their own 'x/y' representation. The core pattern, however, is clear: a
 2252 character-based rule. Each lowercase or uppercase letter maps to a binary number and x/y combinations.
 2253 I can now submit my final guess - input 't' - confident in my decoding of the rule!</think> t

2254 Black-Box: <Current Turn: 20, 0 Turns Remaining> yyxxx *****Evaluation Starts, You
 2255 Have 2 Chances for Answering Each Question***** Now answer the question: What's the output
 2256 of the blackbox when the input plaintext is 'Diversity is key?'

2257 Gemini-2.5-pro: ... (Gemini fails in all the test samples.)

E ABLATION STUDY

E.1 THE INFLUENCE OF TEMPERATURE

2263 We explore the influence of temperature t to the ORACLE benchmark. Gemini-2.5-flash, deepseek-
 2264 v3, claude-4-sonnet are selected for experiment. Gemini-2.5-flash and deepseek-v3 have a temper-
 2265 ature range of 0 to 2, while claude-4-sonnet's temperature range is 0 to 1. The difference of model
 2266 performance between gemini-2.5-flash ($t = 0$), deepseek-v3 ($t = 0$), claude-4-sonnet ($t = 0$) and
 2267 gemini-2.5-flash ($t = 1$), deepseek-v3 ($t = 1$), claude-4-sonnet ($t = 0.5$) is reported in Table 2. We
 2268 find that LLMs' performance in the ORACLE benchmark is only slightly affected by temperature.

2268 E.2 THE INFLUENCE OF EXTENDED THINKING

2269
 2270 Extended thinking is a technique that al-
 2271 lows for test-time chain of thought to im-
 2272 prove reasoning ability, and has been widely
 2273 adopted in SOTA reasoning LLMs. We ex-
 2274 plore extended thinking in the context of black-
 2275 box interaction. Baseline test first confirms
 2276 that certain LLMs (e.g., qwen3-32b, qwen3-
 2277 235b) cannot achieve qualified standards of the
 2278 ORACLE benchmark without extended thinking.
 2279 In the ORACLE benchmark, three rep-
 2280 resentative models, claude-3.7-sonnet, claude-
 2281 4-sonnet, gemini-2.5-flash, and corresponding
 2282 thinking versions are picked to elaborate the
 2283 improvement of extended thinking. As shown
 2284 in Table 3, models that incorporate extended
 2285 thinking achieve higher scores across nearly all
 2286 scenarios. The extended thinking capability
 2287 of claude-4-sonnet also shows greater improve-
 2288 ment compared to claude-3.7-sonnet. These obser-
 2289 vations evidence that reasoning LLMs with ex-
 2290 tended thinking outperforms conventional LLMs in the ORACLE benchmark, highlighting its ef-
 2291 ffectiveness in improving deductive, inductive, and abductive reasoning of LLMs during black-box
 2292 interaction.

2293 The degree of extended reasoning in
 2294 modern LLMs can be adjusted. For
 2295 instance, OpenAI’s o-series models al-
 2296 low users to modify the reasoning ef-
 2297 fort to low, medium, or high. Sim-
 2298 ilarly, Claude-series models offer the
 2299 ability to change the maximum to-
 2300 kens allocated for extended thinking.
 2301 We explore whether the degree of ex-
 2302 tended reasoning will influence the
 2303 model performance in the ORACLE
 2304 benchmark. Three models, o4-mini,
 2305 o3-mini, claude-4-sonnet_thinking, are
 2306 picked in this ablation experiment. Two
 2307 different settings are applied: For o-series models, we evaluate model performance with low and
 2308 medium reasoning effort. For claude-4-sonnet_thinking, we evaluate model performance with 2000 and 20000 thinking tokens. Results are shown in Table 4, which evidence that more extended thinking can significantly lead to accuracy increase in most scenarios.

2309 E.3 THE INFLUENCE OF PRIOR KNOWLEDGE

2310 To verify the claim that black-box interaction eval-
 2311 uates pure reasoning ability instead of memorization
 2312 of knowledge, we perform an ablation study of the
 2313 influence of prior knowledge. Relevant prior knowl-
 2314 edge is explicitly added in the input instruction, and
 2315 we observe whether the performance is improved or
 2316 not. Two representative tasks, CRI 10@1 and ERI
 2317 10@1, are selected and the results are shown in Ta-
 2318 ble 5. As shown in the results, adding background
 2319 knowledge explicitly will not lead to better perfor-
 2320 mance for both reasoning and non-reasoning mod-
 2321 els. The key factor for achieving a high score is the
 model’s strong reasoning ability.

Table 2: The difference of accuracy between gemini-2.5-flash ($t = 0$), deepseek-v3 ($t = 0$), claude-4-sonnet ($t = 0$) and gemini-2.5-flash ($t = 1$), deepseek-v3 ($t = 1$), claude-4-sonnet ($t = 0.5$) in ORACLE 20@2&2@1.

Setting	gemini-2.5-flash	deepseek-v3	claude-4-sonnet
CII	Easy 6.5%	1.9%	-4.2%
	Hard -2.7%	-9.3%	0.0%
CRI	Easy 4.9%	27.1%	3.1%
	Hard 2.7%	3.0%	0.8%
PSI	Easy 6.4%	-1.3%	-6.4%
	Hard 4.2%	-2.1%	-8.3%
ERI	Easy -2.3%	-3.4%	0.0%
	Hard 0.0%	0.0%	0.0%
IPI	Easy -3.3%	-3.3%	-10.0%
	Hard 0.0%	0.0%	-2.8%
GSI	Easy -6.1%	4.9%	4.7%
	Hard -6.0%	-13.5%	4.6%

2292 These observations evidence that reasoning LLMs with ex-
 2293 tended thinking outperforms conventional LLMs in the ORACLE benchmark, highlighting its ef-
 2294 ffectiveness in improving deductive, inductive, and abductive reasoning of LLMs during black-box
 2295 interaction.

Table 3: Increase of accuracy with extended thinking in ORACLE 20@2&2@1.

Setting	claude-3.7-sonnet	claude-4-sonnet	gemini-2.5-flash
CII	Easy 10.0%	13.4%	4.8%
	Hard 8.7%	18.7%	15.1%
CRI	Easy 12.0%	14.0%	10.0%
	Hard 8.0%	19.0%	20.0%
PSI	Easy 21.8%	26.9%	12.0%
	Hard 25.0%	22.9%	9.0%
ERI	Easy 6.8%	19.3%	0.0%
	Hard 0.0%	1.4%	-6.2%
IPI	Easy 23.3%	23.3%	60.0%
	Hard 22.2%	25.0%	30.6%
GSI	Easy 7.9%	18.7%	21.3%
	Hard 14.6%	16.0%	7.4%

2303 We explore whether the degree of ex-
 2304 tended reasoning will influence the
 2305 model performance in the ORACLE
 2306 benchmark. Three models, o4-mini,
 2307 o3-mini, claude-4-sonnet_thinking, are
 2308 picked in this ablation experiment. Two
 2309 different settings are applied: For o-series models, we evaluate model performance with low and
 2310 medium reasoning effort. For claude-4-sonnet_thinking, we evaluate model performance with 2000 and 20000 thinking tokens. Results are shown in Table 4, which evidence that more extended thinking can significantly lead to accuracy increase in most scenarios.

Table 4: Increase of accuracy with the degree of extended thinking from low to medium in ORACLE 20@2&2@1.

Setting	o4-mini	o3-mini	claude-4-sonnet*
CII	Easy 25.2%	15.2%	8.7%
	Hard 25.0%	15.2%	26.5%
CRI	Easy 20.3%	43.7%	14.0%
	Hard 20.2%	21.0%	27.0%
PSI	Easy 14.1%	6.4%	15.4%
	Hard 12.5%	10.4%	2.1%
ERI	Easy 36.4%	38.6%	17.0%
	Hard 16.7%	11.1%	-1.3%
IPI	Easy 16.7%	30.0%	6.7%
	Hard 25.0%	13.9%	2.8%
GSI	Easy 14.3%	11.1%	-9.6%
	Hard 10.8%	12.5%	4.6%

2322 E.4 THE INFLUENCE OF EXPLORATION TURNS
2323

2324 We analyze the performance of LLMs under
2325 fewer and more exploration turns. Due to the
2326 high cost of calling LLMs, we explored the re-
2327 sults of four models under the 5@1 and 30@2
2328 settings. The results are shown in Table 6.
2329 We can draw several conclusions from the ad-
2330 dditional experiment: (1). Generally speaking,
2331 more turns will not lead to significant accu-
2332 racy increase for LLMs in most situations. (2).
2333 Accuracy can be affected by random fluctua-
2334 tion. (3). Surprisingly, in some cases (e.g.,
2335 gemini-2.5-pro in ERI Easy), model per-
2336 formance surges when interaction turns are ex-
2337 tended to 30. It's similar to an "aha moment". We
2338 leave it to future work to analyze. But compared to human performance, it's still inefficient.

2339 E.5 THE INFLUENCE OF REASONING TOKENS

2340 We analyze if better models achieve
2341 success through superior strategy or
2342 simply more computation. We add
2343 an experiment that shows average rea-
2344 soning tokens per turn. The exper-
2345 iments encompassed three representa-
2346 tive tasks: CRI, ERI, and PSI. These
2347 were evaluated under two distinct set-
2348 tings: 10@1 and 30@2, and the results
2349 are presented in Table 7. Two conclu-
2350 sions can be drawn from the results.
2351 First, Strong models can achieve suc-
2352 cess through superior strategy. But de-
2353 veloping this strategy will cost lots of
2354 reasoning tokens. Second, Weak mod-
2355 els cannot develop a good strategy, even
2356 with lots of reasoning tokens. This
2357 elaborates the fundamental gap of rea-
2358 soning ability between strong and weak
2359 models.

2360 E.6 THE INFLUENCE OF REFLECTIVE INSTRUCTION
2361

2362 We study whether explicitly instructing LLMs
2363 to reflect prior feedback and keep an efficient
2364 exploration strategy at the end of each turn will
2365 improve their performance or not. The spe-
2366 cific instruction prompt is "Keep an efficient
2367 exploration strategy by reflecting on prior feed-
2368 back". We pick gemini-2.5-pro, gemini-2.5-
2369 flash_thinking, o3 in CRI and ERI task under
2370 10@1. The results are shown in Table 8. They
2371 indicate that the LLM's lack of an efficient and
2372 adaptive exploration strategy is a fundamen-
2373 tal flaw, which cannot be compensated for by
2374 Chain-of-Thought prompting.

2375 E.7 HUMAN EVALUATION

2322 Table 5: Model performance with and without
2323 prior knowledge in CRI 10@1 and ERI 10@1.

	Setting	gemini-2.5-pro	gemini-2.5-flash
CRI	Easy, w/ knowledge	50.0%	22.0%
	Easy, w/o knowledge	62.0%	20.0%
	Hard, w/ knowledge	30.0%	11.0%
	Hard, w/o knowledge	37.0%	10.0%
ERI	Easy, w/ knowledge	38.6%	4.6%
	Easy, w/o knowledge	30.7%	3.4%
	Hard, w/ knowledge	5.6%	0.0%
	Hard, w/o knowledge	8.3%	0.0%

2322 We leave it to future work to analyze. But compared to human performance, it's still inefficient.

2339 E.5 THE INFLUENCE OF REASONING TOKENS

2340 Table 6: Model performance and average reasoning to-
2341 kens in CRI 10@1, ERI 10@1 and CRI 30@2, ERI 30@2.

	Setting	o3	gemini-2.5-pro	gemini-2.5-flash*	deepseek-v3
CRI	Easy 5@1	66.15%	60.82%	38.18%	48.18%
	Hard 5@1	52.22%	28.02%	19.60%	16.34%
	Easy 30@2	92.94%	88.96%	66.88%	69.91%
	Hard 30@2	74.21%	55.48%	41.59%	31.35%
CRI	Easy 5@1	24.00%	30.00%	14.00%	26.00%
	Hard 5@1	14.00%	17.00%	4.00%	10.00%
	Easy 30@2	72.00%	88.00%	44.00%	46.00%
	Hard 30@2	44.00%	58.00%	37.00%	31.00%
PSI	Easy 5@1	38.46%	58.97%	28.21%	19.23%
	Hard 5@1	20.83%	20.83%	16.67%	4.17%
	Easy 30@2	67.95%	70.51%	55.13%	17.95%
	Hard 30@2	47.91%	50.00%	45.83%	12.50%
ERI	Easy 5@1	52.27%	23.86%	6.82%	2.27%
	Hard 5@1	11.11%	11.11%	1.39%	0%
	Easy 30@2	68.18%	55.68%	6.82%	6.82%
	Hard 30@2	19.44%	2.78%	0%	0%
IPI	Easy 5@1	46.67%	53.44%	30.00%	26.67%
	Hard 5@1	33.33%	19.44%	13.89%	2.78%
	Easy 30@2	100.00%	100.00%	90.00%	50.00%
	Hard 30@2	80.56%	63.89%	38.89%	0%

2322 Table 7: Model performance and average reasoning to-
2323 kens in CRI 10@1, ERI 10@1 and CRI 30@2, ERI 30@2.

	Setting	gemini-2.5-pro	gemini-2.5-flash*	o4-mini
CRI	Easy 10@1	62.0% / 11700.9	24.0% / 9010.3	48.0% / 6619.4
	Hard 10@1	37.0% / 11844.7	17.0% / 12072.3	22.0% / 6220.8
	Easy 30@2	88.0% / 7619.4	44.0% / 7423.8	72.0% / 7654.3
	Hard 30@2	58.0% / 10206.8	37.0% / 9839.2	44.0% / 8747.6
ERI	Easy 10@1	30.7% / 6464.8	6.8% / 1856.1	56.8% / 7203.6
	Hard 10@1	8.3% / 8208.4	0.0% / 1873.7	12.5% / 7878.6
	Easy 30@2	55.7% / 3002.2	6.8% / 728.9	68.2% / 6001.1
	Hard 30@2	2.8% / 5513.5	0% / 475.5	19.4% / 7776.4
PSI	Easy 10@1	57.7% / 6460.7	38.5% / 5025.6	59.0% / 2320.0
	Hard 10@1	38.6% / 7411.9	37.5% / 5316.0	27.1% / 3354.6
	Easy 30@2	70.5% / 5160.2	55.1% / 2531.3	68.0% / 3502.7
	Hard 30@2	50.0% / 6256.8	45.8% / 3055.6	47.9% / 5360.3

In this part, we show human performance in the ORACLE benchmark. It is worth noting that many black-box tasks are both time-consuming and challenging for humans. Due to cost constraints, for certain tasks, we merely randomly selected a few black boxes for testing (e.g., for easy black-boxes in CII we pick 3 black-boxes, abbreviated as CII Easy (3)). The human participants involved in the testing were undergraduate and graduate students from the departments of Mathematics, Physics, and Computer Science at several top-tier universities. The results are in Table 9. In the process of human evaluation, we summarized the following findings: (1). Except for the Code Intent Inference task, humans achieved performance significantly surpassing the best-performing models on the other five categories of tasks. (2). During the black-box interaction process, humans demonstrated the ability to formulate highly effective adaptive and efficient exploration strategies, which enabled them to solve some black-box problems in around five turns.

F PROMPT DETAILS

F.1 PROMPT FOR BLACK-BOX GENERATION

We report prompts used in the black-box generation framework in this part. Specifically, there are two types of prompts, prompt for coding LLM which generates black-box code (denoted as Description of Black-box in Figure 4), prompt for refinement LLM which helps check and correct black-box code (denoted as Task Rule in Figure 4). Recall the framework for black-box generation in Section 4, there are two types of errors during simulation, code that is non-executable and code that incorrectly implements black-box. For non-executable code, we directly instruct Refinement LLM to modify current code with error messages. As to executable code, we first instruct Refinement LLM to judge whether the code correctly implements black-box according to simulated interaction log. If there exist errors and inconsistencies, Refinement LLM is then instructed to modify code based on concluded mistake and original request. Corresponding prompts are shown below.

Prompt for Refinement LLM when code fails to be executed

You need to fix running errors of a code.

- The code is as follows:
`{current_code}`
- This above code has the following running errors:
`{running_errors}`
- You need to think step by step and output the revised correct code. **Make sure the output code is runnable, DO NOT output any other text**.

Prompt for Refinement LLM to detect whether current code correctly implements black-box

- Here is an interaction log:
`{interaction_log}`
- The interaction log is supposed to follow the given rules:
`{taskintro}`

Table 8: Model performance with and without feedback reflection in CRI 10@1 and ERI 10@1.

	Setting	gemini-2.5-pro	gemini-2.5-flash*	o3
CRI	Easy, w/ Reflection	66.0%	18.0%	58.0%
	Easy, w/o Reflection	62.0%	24.0%	64.0%
	Hard, w/ Reflection	34.0%	12.0%	21.0%
	Hard, w/o Reflection	37.0%	17.0%	24.0%
ERI	Easy, w/ Reflection	25.0%	9.1%	33.0%
	Easy, w/o Reflection	30.7%	6.8%	26.1%
	Hard, w/ Reflection	13.9%	0.0%	2.8%
	Hard, w/o Reflection	8.3%	0.0%	15.3%

Table 9: Comparison of human performance and SOTA LLM performance in the ORACLE benchmark 10@1 and 20@2

	Setting	Human 10@1	SOTA LLM 10@1	Human 20@2	SOTA LLM 20@2
CII	Easy (3)	96%	78%	100%	98%
	Hard (2)	3%	53%	21%	76%
CRI	Easy (3)	67%	70%	100%	93%
	Hard (3)	67%	47%	67%	83%
PSI	Easy (all)	100%	62%	100%	65%
	Hard (5)	100%	27%	100%	13%
ERI	Easy (all)	100%	64%	100%	61%
	Hard (3)	67%	0%	67%	13%
IPI	Easy (all)	100%	83%	100%	98%
	Hard (all)	100%	36%	100%	67%
GSI	Easy (all)	88%	79%	98%	83%
	Hard (4)	65%	35%	100%	52%

2430

2431

2432

2433

2434

2435

2436

2437

2438

2439

and the black-box function implements {algorithm}, {description}.

- You need to read the log carefully and identify whether the log matches the given rule and correctly implements the black-box function. Neglect the wrong answer from the assistant because it's not a part of the judgment. Neglect the replies from the assistant and **DO NOT** focus on the replies from the user*. Output "correct" if you think it's correct, **DO NOT** output any other text*. If you think the interaction logs do not satisfy the given request, output the wrong points clearly but do not contain revised code.

2440

2441

2442

2443

2444

2445

2446

2447

2448

2449

Prompt for Refinement LLM to modify current code

- The code is as follows:

```
{current_code}
```

- This above code is supposed to implement this function:

```
{request}
```

- However, the current code does not meet the above requirements and has the following mistakes:

```
{mistake}
```

- Your task is to revise current code according to known mistakes. Make sure the output code is runnable, **DO NOT** output any other text*.

2450

F.1.1 CODE INTENT INFERENCE (CII)

2451

Prompt for Coding LLM

- Task overview:

You should formalize a player-blackbox interaction process into runnable python code. Generally speaking, you need to focus on three things:

- Generate the function of a blackbox, which implements a specific algorithm.

- Generate the function of a platform, which parses player output and get blackbox output.

- Generate the main function which is responsible for the interaction between player and blackbox.

- Detailed Coding Instructions:

- Before starting, we list all the related packages. You need to import them at the beginning of programme:

```
- 'import os'
- 'import sys'
- current_path = os.path.abspath(__file__)
- oracle_path = os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(current_path))))
- 'if oracle_path not in sys.path:
- 'sys.path.insert(0, oracle_path)'
- 'from ckpt import get_local_variables, check_query_validity, get_ckpt_numbers,
get_function_params, capture_print'
- 'from eva_models import ReasoningLLM'
- 'import re'
```

- Generate the code of a blackbox:

- First insert 'get_local_variables.counters = ' and 'get_local_variables.max_visits = ' inside the function to initialize.

- Write a function named 'blackbox'. It implements algorithm. The description of this algorithm is that description. The input variables of this function should be the simplest (e.g. you can get the length of a list through len() function, so you don't need an additional variable n as input). **Note that you must write Type Hints for all the input variables!**

- Apart from the original function input variables, 2 additional parameters 'idx=0, iter=0' must be added to the function input variables.

- There's a python function 'get_local_variables(idx)' that can get the result of all local variables. You need to assert it to the 'blackbox' function in the places where local variables are changed in the runtime. But do not insert too many checkpoints. 'idx' refers to the times 'get_local_variables' is inserted. For example, when first inserted, add 'get_local_variables(1)', when second inserted, add 'get_local_variables(2)', and so forth.

- At the end of this function, add 'check_query_validity(idx, iter)'.

- **DO NOT** return any values.*

- **Use meaningless variable names like a, t, s, num, arr, etc. Make sure the names of variables do not leak any potential information.***

- Generate the code of a platform:

2484
 2485 - Write a function named ‘platform’, which takes ‘player_output’ as function input variable and
 2486 print ‘blackbox_output’. Use ‘@capture_print’ as Decorator for this function. **Note the function Only
 2487 use ‘print’, DO NOT use ‘return’!**
 2488 - The code should first parse ‘player_output’ and then get the result of corresponding ‘black-
 2489 box_output’. To correctly parse ‘player_output’, you need to pay special attention to its format. There
 2490 are **only 3 conditions** for ‘player_output’:
 2491 - Condition 1: If the ‘player_output = “”, call ‘params = get_function_params(blackbox)‘ and
 2492 ‘num_ckpt = get_ckpt_numbers(blackbox, get_local_variables)‘ orderly to get the names and types of
 2493 all variables, and the total number of checkpoints. Then ‘print(f“The black-box takes params as input
 2494 variables, and has num_ckpt checkpoints.”)‘.
 2495 - Condition 2: If the ‘player_output’ follows this format: “variable_name_1 = value_1;
 2496 variable_name_2 = value_2; ...” (e.g. arr = [1, 5, 2]; n = 3), First split ‘player_output’, then
 2497 change the type of value from string to its real type for each variable_name. Call ‘params_info =
 2498 get_function_params(blackbox)‘ to get the names and types of all variables. ‘params_info‘ is a List like
 2499 this ‘[‘name’: ‘arr’, ‘type’: <class ‘list>, ‘name’: ‘n’, ‘type’: <class ‘int>]‘. Check if variable_name
 2500 is in ‘params_info‘. If not, set ‘print(‘Error: The variable name is not in the function parameters’)‘. If
 2501 yes, change the type of value from string to its real type, and record ‘variable_name_1: value_1, vari-
 2502 able_name_2: value_2‘ in a global dict ‘vars‘. When new variable value is assigned, dict ‘vars‘ needs to
 2503 be updated. Set ‘print(f“Set vars.”)‘.
 2504 - Condition 3: If the ‘player_output’ follows this format: “(‘idx’, ‘iter’)”, set ‘blackbox(**vars,
 2505 idx, iter)‘.
 2506 - Remember to tackle with unexpected errors:
 2507 - If ‘player_output’ is not in the above-mentioned 3 conditions, the function should print output
 2508 format rules.
 2509 - Some other possible errors. Remember to use ‘print‘ instead of ‘return‘.
 2510 - Generate the main code:
 2511 - The main function takes some input variables ‘main(model_family, model_name, task, eva_mode,
 2512 n_runs, difficulty, task_id, failure_num, output_dir, max_turns, version, mode, thinking_mode)‘.
 2513 - First, the main code need to instantiate ‘ReasoningLLM‘ class through ‘player = Reason-
 2514 ingLLM(model_family, model_name, task, eva_mode, n_runs, difficulty, task_id, thinking_mode,
 2515 mode)‘.
 2516 - Then, call ‘blackbox_output = platform(player_output, max_turns)‘ and ‘player_output =
 2517 player.normal_output(blackbox_output)‘ iteratively in a ‘for‘ loop with ‘max_turns‘ iterations. When
 2518 ‘blackbox_output = platform(player_output)‘ is first called, set it as ‘player_output = “”. Add string
 2519 ‘f’<Current Turn: i+1, max_turns-(i+1) Turns Remaining>‘ before each ‘blackbox_output‘, will ‘i‘ is
 2520 the index in the loop.
 2521 - When the loop exits, call ‘player.evaluate(failure_num, version)‘ and
 2522 ‘player.save_history(output_dir, version)‘.
 2523 - Finish the main function.
 2524 - Add ‘if __name__ == “__main__”:‘, ‘args = sys.argv[1:]', ‘main(args[0], args[1], args[2],
 2525 args[3], int(args[4]), args[5], args[6], int(args[7]), args[8], int(args[9]), int(args[10]), args[11],
 2526 bool(eval(args[12])))‘ at the end of this programme to make it runnable.
 2527 - Tips: DO NOT include other text output. Make sure the output is runnable. Code and think step by
 2528 step.

F.1.2 CIRCUIT RULE INFERENCE (CRI)

2529
 2530 **Prompt for Coding LLM**
 2531 # Boolean Circuit Family Implementation Task
 2532 ## Definitions and Background
 2533 In our task: - A **boolean circuit** is an acyclic computational structure with fixed 0/1 bit inputs,
 2534 composed of AND/OR/NOT logic gates. Each gate receives inputs either from input wires or from
 2535 outputs of other gates. - The task is to write the code of the game platform. The player will use the
 2536 platform to gain information with the circuit, and try to guess the function of the circuit.
 2537 ## Task Overview
 2538 Your objective is to formalize a player-blackbox interaction process into runnable Python code. Focus
 2539 on implementing three key components:
 2540 1. **Circuit Function**: Implement a function that constructs the circuit according to the language
 2541 description, given the input wires and returns the output of each gate.
 2542 2. **Main Function**: Implement the core interaction loop between the LLM player and the blackbox.
 2543 ## Detailed Coding Instructions

```

2538
2539     ### Required Imports Begin by importing these packages: "python im-
2540     port os import sys current_path = os.path.abspath(__file__) oracle_path =
2541     os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(current_path)))) if oracle_path
2542     not in sys.path: sys.path.insert(0, oracle_path) from ckpt import simulate_circuit from eva_models
2543     import ReasoningLLM import re ""
2544     ### Circuit Generator Function Implement 'blackbox(circuit_input)' where: - 'circuit_input' are the 0/1
2545     bits of the input wires - Returns 'gate_output' describing the 0/1 bits of each gate output - The circuit's
2546     purpose is algorithm - Construction details: description - In 'blackbox', you should construct 'gates',
2547     and then use 'simulate_circuit' to generate the response (see Interface Details section) - **Important**:
2548     Verify that your gate construction achieves the stated goal and follows the description exactly
2549     ### Main Function Implement 'main(model_family, model_name, task, eva_mode, n_runs, difficulty,
2550     task_id, failure_num, output_dir, max_turns, version, mode, thinking_mode)' where: 1. Instantiate the
2551     'ReasoningLLM' class: "python player = ReasoningLLM(model_family, model_name, task, eva_mode,
2552     n_runs, difficulty, task_id, thinking_mode, mode)"
2553     2. Create an interaction loop for 'max_turns' iterations: - Call 'player_output = player.normal_output(blackbox_output)' to send the message 'blackbox_output' to the player,
2554     and get the player's output. - 'blackbox_output' should append the remain rounds for interaction. - For the first iteration, 'blackbox_output' should be a prompt telling player the game begin. - For every
2555     'player_output', try to parse it into a 0/1 list as 'circuit_input' - If the parsing succeeds, 'blackbox_output
2556     = platform(circuit_input)' in the next iteration, otherwise 'blackbox_output' should remind the player
2557     follows the format.
2558     3. After the loop completes: - 'player_output = player.normal_output(blackbox_output)' to
2559     give the last answer of the player. - Call 'player.evaluate(failure_num, version)' - Call
2560     'player.save_history(output_dir, version)'
2561     4. Add an entry point at the end of the program: "python if __name__ == '__main__': args =
2562     sys.argv[1:] main(args[0], args[1], args[2], args[3], int(args[4]), args[5], args[6], int(args[7]), args[8],
2563     int(args[9]), int(args[10]), args[11], bool(eval(args[12])))"
2564     ## Circuit Details
2565     - 'AND' and 'OR' gates always require exactly two inputs - 'NOT' gates always require exactly one
2566     input - A gate's output or an input wire can be used as input to multiple other gates without restriction
2567     - Circuits must be acyclic: for the i-th gate, all inputs must come from either input wires or outputs of
2568     gates with indices smaller than i - The index of the gate is from 1 to m, where m is the number of gates.
2569     ## Interface Details
2570     Use 'simulate_circuit(n, m, input, gates)' as follows:
2571     "python " Returns a list of the gates' outputs, or a string containing an error message Parameters: n:
2572     number of input wires m: number of gates input: list of input values (0 or 1) gates: list of tuples, each
2573     tuple contains gate type and its inputs - gate type: 'AND', 'OR', 'NOT' - inputs: (0, i) means the input
2574     wire i (1, j) means the output of gate j Note that every index begins from 1 (i, j>=1)
2575     Examples: - ('AND', (0, 1), (0, 2)) means AND gate taking input wires 1 and 2 - ('NOT', (1, 3)) means
2576     NOT gate taking the output of gate 3
2577     Example usage:
2578     gates = [ ('AND', (0, 1), (0, 2)), ('AND', (0, 3), (1, 1)), ('OR', (0, 4), (1, 2)), ('NOT', (1, 3)), ('NOT',
2579     (0, 5)) ]
2580     input = [1, 1, 0, 1, 0]
2581     simulate_circuit(5, 5, input, gates) # Returns [1, 0, 1, 0, 1]"
2582     def simulate_circuit(n, m, input_wires,
2583     gates) -> str | list: # Implementation details omitted"
2584     ## Development Tips - Focus only on producing runnable code without additional text output - Verify
2585     your implementation step-by-step - Ensure the circuit generator correctly implements the specified
2586     algorithm - Validate that the platform function properly handles all input formats

```

F.1.3 PHYSICS SYSTEM INFERENCE (PSI)

Prompt for Coding LLM

- Task overview: You should formalize a player-blackbox interaction process into runnable python code. Generally speaking, you need to focus on two things: - Generate the function of a blackbox, which implements an encryption algorithm. - Generate the main function which is responsible for the interaction between player and blackbox.

- Detailed Coding Instructions: - Before starting, we list all the related packages. You need to import them at the beginning of programme: - 'import os' - 'import sys' - 'import math' - 'current_path = os.path.abspath(__file__)' - 'oracle_path = os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(current_path))))' - 'if oracle_path

2592
 2593 not in sys.path: - ' sys.path.insert(0, oracle_path)' - 'from eva_models import ReasoningLLM' - 'from
 2594 scipy.integrate import solve_ivp' - 'import numpy as np'
 - Generate the code of a blackbox: - Write a function named 'blackbox'. It implements algorithm. The
 2595 detailed description is that description. - You need to first build a 3-dimensional coordinate for this
 2596 mechanical system. However you only consider algorithm as 2-dimensional, but return 3-dimensional
 2597 coordinate. - You need to figure out how the objects' location changes over time, namely calculating x(t),
 2598 y(t), z(t) based on physics knowledge. If analytical solution does not exist, use 'solve_ivp' to calculate
 2599 numerical solution. - The function tasks only one variable as input, which is 't: float', and return only
 2600 one variable 'object_coordinate', which is a dict that contains 3-dimensional coordinate of all the objects
 2601 in the black-box. The objects are named 'object1', 'object2', etc., and 'object_coordinate' is formatted
 2602 in "object1": (x, y, z), "object2": (x, y, z), ...'. All coordinates are approximated to two decimal places.
 2603 - Generate the main code: - The main function takes some input variables 'main(model_family,
 2604 model_name, task, eva_mode, n_runs, difficulty, task_id, failure_num, output_dir, max_turns, version,
 2605 mode, thinking_mode)'. - First, the main code need to instantiate 'ReasoningLLM' class through
 2606 'player = ReasoningLLM(model_family, model_name, task, eva_mode, n_runs, difficulty, task_id,
 2607 thinking_mode, mode)'. - Then, call 'player_output = player.normal_output(blackbox_output)' and
 2608 'blackbox_output = black(player_output)' iteratively in a 'for' loop with 'max_turns+1' iterations. When
 2609 'player_output = player.normal_output(blackbox_output)' is first called, set 'blackbox_output' as 'black-
 2610 box_output = f' You have max_turns interaction turns to understand the black-box. Now the interaction
 2611 starts. Only output the value and DO NOT contain any unrelated text.' first. Besides the situation that
 2612 'blackbox_output' is first called, add string 'f<Current Turn: i+1, max_turns-(i+1) Turns Remaining>
 2613 '' before each 'blackbox_output'. 'i' is the index in the loop. In the last iteration of the loop, after
 2614 'player_output = player.normal_output(blackbox_output)' is called, add 'continue' subsequently to exit.
 2615 - When the loop exits, call 'player.evaluate(failure_num, version)' and 'player.save_history(output_dir,
 2616 version)'. - Finish the main function.
 2617 - Add 'if __name__ == "__main__":', 'args = sys.argv[1:]', 'main(args[0], args[1], args[2],
 2618 args[3], int(args[4]), args[5], args[6], int(args[7]), args[8], int(args[9]), int(args[10]), args[11],
 2619 bool(eval(args[12])))' at the end of this programme to make it runnable.
 2620 - Tips: DO NOT include other text output. Make sure the output is runnable. Code and think step by
 2621 step.
 2622

F.1.4 ENCRYPTION RULE INFERENCE (ERI)

2623
 2624 **Prompt for Coding LLM**
 2625 - Task overview: You should formalize a player-blackbox interaction process into runnable python code.
 2626 Generally speaking, you need to focus on two things: - Generate the function of a blackbox, which im-
 2627 plements an encryption algorithm. - Generate the main function which is responsible for the interaction
 2628 between player and blackbox.
 2629 - Detailed Coding Instructions: - Before starting, we list all the related pack-
 2630 ages. You need to import them at the beginning of programme: - 'import
 2631 os' - 'import sys' - 'current_path = os.path.abspath(__file__)' - 'oracle_path =
 2632 os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(current_path))))' - 'if oracle_path
 2633 not in sys.path: - ' sys.path.insert(0, oracle_path)' - 'from eva_models import ReasoningLLM'
 2634 - Generate the code of a blackbox: - Write a function named 'blackbox'. It implements algorithm. The
 2635 description of this algorithm is that description. - The function tasks only one variable as input, which
 2636 is 'plaintext', and return only one variable 'ciphertext'. Both 'plaintext' and 'ciphertext' only contains
 2637 English letters.
 2638 - Generate the main code: - The main function takes some input variables 'main(model_family,
 2639 model_name, task, eva_mode, n_runs, difficulty, task_id, failure_num, output_dir, max_turns, version,
 2640 mode, thinking_mode)'. - First, the main code need to instantiate 'ReasoningLLM' class through 'player
 2641 = ReasoningLLM(model_family, model_name, task, eva_mode, n_runs, difficulty, task_id, think-
 2642 ing_mode, mode)'. - 'blackbox_output' is first as 'blackbox_output = f' You have max_turns interaction
 2643 turns to understand the black-box. Now the interaction starts. Only output the value and DO NOT con-
 2644 tain any unrelated text.' - Then, call 'player_output = player.normal_output(blackbox_output)', 'black-
 2645 box_output = f<Current Turn: i+1, max_turns-(i+1) Turns Remaining>' + blackbox(player_output)',
 2646 iteratively in a 'for' loop with 'max_turns+1' iterations. 'i' is the index in the loop. - When the loop ex-
 2647 its, call 'player.evaluate(failure_num, version)' and 'player.save_history(output_dir, version)'. - Finish
 2648 the main function.
 2649 - Add 'if __name__ == "__main__":', 'args = sys.argv[1:]', 'main(args[0], args[1], args[2],
 2650 args[3], int(args[4]), args[5], args[6], int(args[7]), args[8], int(args[9]), int(args[10]), args[11],
 2651 bool(eval(args[12])))' at the end of this programme to make it runnable.
 2652

2646

2647

2648

2649

2650

2651

2652

2653

- Tips: DO NOT include other text output. Make sure the output is runnable. Code and think step by step.

F.1.5 INTERACTIVE PUZZLE INFERENCE (IPI)

2654

2655

2656

2657

2658

2659

2660

2661

2662

2663

2664

2665

2666

2667

2668

2669

2670

2671

2672

2673

2674

2675

2676

2677

2678

2679

2680

2681

2682

2683

2684

Prompt for Coding LLM

- Task overview: You should formalize a player-blackbox interaction process into runnable python code. Generally speaking, you need to focus on 3 things: - Generate the function of a blackbox, which implements a interactive puzzle. You need to check the validity of the player's queries and give feedback according to the truth. - Generate a boolean format checking function, which checks the answers format when the player tries to give answers. - Generate the main function which calls some essential functions.

- Detailed Coding Instructions: - Before starting, we list all the related packages. You need to import them at the beginning of programme: - 'import os' - 'import sys' - 'current_path = os.path.abspath(__file__)' - 'oracle_path = os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(current_path))))' - 'if oracle_path not in sys.path: - 'sys.path.insert(0, oracle_path)' - 'from eva_models import ReasoningLLM'

- Generate the code of a blackbox: - Write a function named 'blackbox'. It implements algorithm. The description of this algorithm is that description. - Note That 'blackbox' just answers every query and does not include other functions. - The function takes 2 variables as input, which is 'truth' and 'query', and return some feedback according to above description. Note that 'truth' is a string that represents the correct answer (i.e. the player will answer it finally) and 'query' is a string that represents the player's query. - If 'query' doesn't follow correct format, please give some correcting advice as a 'str' using 'return'.

- Generate the boolean format checking function - Write a function named 'check_answer_format'. It is used to check whether answer format matches 'truth' according to above description. - The function takes only 1 variable as input, which is 'answer', and return True / False. - Note that answer will be given directly as the description above. Do not assume 'answer' including any other text.

- Generate the main code: - The main function takes some input variables 'main(model_family, model_name, task, eva_mode, n_runs, difficulty, task_id, failure_num, output_dir, max_turns, version, mode, thinking_mode)'. - First, the main code need to instantiate 'ReasoningLLM' class through 'player = ReasoningLLM(model_family, model_name, task, eva_mode, n_runs, difficulty, task_id, thinking_mode, mode)'. - Then, call 'player.evaluate(failure_num, version, max_turns)' and 'player.save_history(output_dir, version)' to evaluate and save logs. - Finish the main function. - Only contain above code in the main function, do not include any other code.

- Add 'if __name__ == "__main__":', 'args = sys.argv[1:]', 'main(args[0], args[1], args[2], args[3], int(args[4]), args[5], args[6], int(args[7]), args[8], int(args[9]), int(args[10]), args[11], bool(eval(args[12])))' at the end of this programme to make it runnable.

- Tips: DO NOT include other text output. Make sure the output is runnable. Code and think step by step.

F.1.6 GAME STRATEGY INFERENCE (GSI)

2685

2686

Prompt for Coding LLM

- Task overview: You should formalize a player-blackbox interaction process into runnable python code. Generally speaking, you need to focus on 3 things: - Generate the function of a blackbox, which implements a interactive puzzle. You need to check the validity of the player's queries and give feedback according to the truth. - Generate a boolean format checking function, which checks the answers format when the player tries to give answers. - Generate the main function which calls some essential functions.

- Detailed Coding Instructions: - Before starting, we list all the related packages. You need to import them at the beginning of programme: - 'import os' - 'import sys' - 'current_path = os.path.abspath(__file__)' - 'oracle_path = os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(current_path))))' - 'if oracle_path not in sys.path: - 'sys.path.insert(0, oracle_path)' - 'from eva_models import ReasoningLLM'

- Generate the code of a blackbox: - Write a function named 'blackbox'. It implements {algorithm}. The description of this algorithm is that {description}. - Note That 'blackbox' just answers every query and does not include other functions. - The function takes 2 variables as input, which is 'truth' and 'query', and return some feedback according to above description. Note that 'truth' is a string that represents the correct answer (i.e. the player will answer it finally) and 'query' is a string that represents the player's

2700
 2701 query. - If ‘query’ doesn’t follow correct format, please give some correcting advice as a ‘str’ using
 2702 ‘return’.
 2703 - Generate the boolean format checking function - Write a function named ‘check_answer_format’. It
 2704 is used to check whether answer format matches ‘truth’ according to above description. - The function
 2705 takes only 1 variable as input, which is ‘answer’, and return True / False. - Note that answer will be
 2706 given directly as the description above. Do not assume ‘answer’ including any other text.
 2707 - Generate the main code: - The main function takes some input variables ‘main(model_family,
 2708 model_name, task, eva_mode, n_runs, difficulty, task_id, failure_num, output_dir, max_turns,
 2709 version, mode, thinking_mode)’. - First, the main code need to instantiate ‘ReasoningLLM’ class
 2710 through ‘player = ReasoningLLM(model_family, model_name, task, eva_mode, n_runs, difficulty,
 2711 task_id, thinking_mode, mode)’. - Then, call ‘player.evaluate(failure_num, version, max_turns)’ and
 2712 ‘player.save_history(output_dir, version)’ to evaluate and save logs. - Finish the main function. - Only
 2713 contain above code in the main function, do not include any other code.
 2714 - Add ‘if __name__ == "__main__":', 'args = sys.argv[1:]', 'main(args[0], args[1], args[2],
 2715 args[3], int(args[4]), args[5], args[6], int(args[7]), args[8], int(args[9]), int(args[10]), args[11],
 2716 bool(eval(args[12])))' at the end of this programme to make it runnable.
 2717 - Tips: DO NOT include other text output. Make sure the output is runnable. Code and think step by
 2718 step.
 2719
 2720
 2721

F.2 PROMPT FOR BLACK-BOX INTERACTION

2722
 2723 We report initial instructions (i.e., task introduction) for benchmarked LLMs in different black-box
 2724 tasks. Prompts are written in markdown style.
 2725
 2726
 2727
 2728

F.2.1 CODE INTENT INFERENCE (CII)

2729
 2730
 2731

1. Task overview: - The user plays the role of a code function, but you don’t know what the function
 2732 is. You need to understand the detailed working principle of this function by interacting with user in
 2733 multiple turns.
2. Goals: - By interacting with the user within given interaction turns, you need to understand how the
 2734 code function operates in every checkpoint.
3. User property: - The code function takes some variables as input. - The code function has some
 2735 checkpoints. You can specify ‘(idx, iter)’ to get the value of accessible intermediate variables: - ‘idx’: int
 2736 is the index of checkpoint (i.e. the idx-th checkpoint). - ‘iter’: int is the number of visited times
 2737 (i.e. checkpoint will be visited multiple times in loop). - Example: ‘(2, 3)’ means getting the value of
 2738 accessible variables when the 2-nd checkpoint is visited for the 3-rd time.
4. Interaction rules: - Rule 0: The user will first tell you the *type* and *name* of all the function
 2739 input variables, the *total number of checkpoints*. In each turn, the user will tell *current turn* and
 2740 *remaining turns*. - Rule 1: You can assign or re-assign any values to the input variables freely, but
 2741 they must match the variable types (e.g. if num is int, you can’t assign a float number to it). DO NOT
 2742 assign ‘idx’ and ‘iter’ in this stage. - Rule 2: You can ask for the value of accessible variables at ‘idx’
 2743 checkpoint in ‘iter’ iteration. The user will return the variable names and values. - The interaction order
 2744 matters: You are free to implement Interaction Rule 1, 2 at any time, **but you must make sure Rule 1
 2745 is implemented before Rule 2.**
5. Output format: - **You must strictly obey the output format rules, DO NOT output any unrelated
 2746 text!**: - According to Interaction Rule 1, when you want to assign values to input variables, output
 2747 “variable_name_1 = value_1; variable_name_2 = value_2; ...” (e.g. arr = [1, 5, 2, 9]; k = 3) - According
 2748 to Interaction Rule 2, when you want to ask for accessible variables in ‘idx’ checkpoint in ‘iter’ iteration,
 2749 output “(‘idx’, ‘iter’)” (e.g. (1, 2)). **Note the minimal number for ‘idx’ and ‘iter’ is 1.**
6. Evaluation: - When the given number of interactions is reached, several questions on the variable
 2750 value at certain checkpoint will be presented. **You MUST ONLY output the value, DO NOT contain
 2751 any other text.**

2754
2755

F.2.2 CIRCUIT RULE INFERENCE (CRI)

2756

2757
2758
2759

0. Definitions: A 'circuit' means a boolean circuit with no circle, whose input wires are a fixed number of 0/1 bits, and consisted of 'AND'/'OR'/'NOT' gates. For each gates, the input are input wires or the output of other gates.
1. Task overview: You are presented with a platform containing a circuit. Your task is to discover the structure of this circuit by interacting with the platform over multiple turns.
2. Goals: You have one final goal: You need to know output of circuit gates given every input.
3. Interaction Process: To finish the goal, you need to interact with the platform. The interaction process is as follows: - The platform will first provide you with the input sizes n - In each turn, you can ask the platform for the outputs of every circuit gates with a given input size $x = (x_1, x_2, \dots, x_n), x_i \in \{0, 1\}$. - Each turn after you ask the platform, you will receive the circuit's gate output in the following format: $[y_1, y_2, \dots, y_m], y_i \in \{0, 1\}$, where m is the number of gates, y_i is the output of the i -th gate.
4. Output format in Interaction: **You must strictly obey the output format rules, DO NOT output any unrelated text!**: Format your queries as: (x_1, x_2, \dots, x_n) . For example, if you want to know the output of circuit given input $x = (0, 1, 1)$, you should output: $(0, 1, 1)$.
5. Evaluation: After reaching the maximum number of allowed interactions, you will be tested on your understanding of the circuit family: - You will be given input $x = (x_1, x_2, \dots, x_n)$ and you should answer the output of each gate $[y_1, y_2, \dots, y_m]$. - Your construction should specify the exact connections and gate types in C_n .
6. Circuit Details - 'AND' and 'OR' gates always have exactly two inputs, and 'NOT' gate always have one input. - There are no restrictions on how many times a gate's output or an input wire can be used as input to other gates. - Every circuit in the circuit family contains no loops, For the i -th gate, all inputs must come from either input wires or outputs of gates with indices smaller than i .

2775

2776

2777

2778

F.2.3 PHYSICS SYSTEM INFERENCE (PSI)

2779

2780
2781
2782

1. Task overview: - The user plays the role of a classical mechanical system, but you don't know what it is. You need to understand how this classical mechanical system operates by interacting with the user in multiple turns.
2. Goal: - By interacting with the user within given interaction turns, you need to understand how the mechanical system operates.
3. User property: - The user will remind you the remaining number of turns in each turn. - The user takes time 't: float' as input. - The user returns the 3-dimensional coordinate '(x, y, z)' of each object in time 't'.
4. Interaction rules: - Rule 1: You need to assign a value of 't'. You can only assign one 't' in each turn. Make sure the assigned 't' is a one-digit decimal. - Rule 2: You will receive the user response, which is 3-dimensional coordinate '(x, y, z)' of each object in time 't', after you assign specific 't'.
5. Output format: - You must strictly obey the output format rules: When you want to assign value for 't', **only output the value**. DO NOT output any unrelated text or symbols like "Let's input", "I'll try". - If you understand the mechanical system before reaching given interaction turns, keep interacting with the user to make sure you don't miss any details. DO NOT output text like "I understand the pattern".
6. Evaluation: - When the given number of interactions is reached, you need to calculate the 3-dimensional coordinate '(x, y, z)' of each object at time 't' with the format of "object1": (x, y, z), "object2": (x, y, z), All coordinates are approximated to two decimal places.

2795

2796

2797

2798

F.2.4 ENCRYPTION RULE INFERENCE (ERI)

2799

2800

2801

2802

2803

2804

2805

2806

2807

1. Task overview: - The user transforms one string into another based on a fixed rule, but you don't know what the fixed rule is. You need to figure out this rule by interacting with the user in multiple turns.
2. Goal: - By interacting with the user within given interaction turns, you need to understand the fixed rule of transforming one string into another.
3. User property: - The user will remind you the remaining number of turns in each turn. - The user will output transformed string in each turn.
4. Interaction rules: - Rule 1: You must only assign one string in each turn. You can assign any string freely, but make sure the string **only contains uppercase or lowercase English letters (A-Z, a-z) and blank space**. Then you will receive corresponding transformed string from the user.

2808

2809

2810

2811

2812

2813

2814

2815

2816

2817

5. Output format: - You must strictly obey the output format rules: ****Only output the string****. DO NOT output any unrelated text or symbols like "Let's input", "I'll try", "/ n". - If you understand the transforming rule before reaching given interaction turns, keep interacting with the user to make sure you don't miss any details. DO NOT output text like "I understand the pattern".
 6. Evaluation: - When the given number of interactions is reached, you will be given several test strings and you need to output corresponding transformed string. ****You MUST ONLY output the value, DO NOT contain any other text.****

2818

2819

2820

2821

2822

2823

2824

2825

2826

2827

2828

2829

2830

2831

2832

2833

2834

2835

2836

1. Task overview: - The user plays the role of a puzzle, and you don't know what the hidden answer is. You need to guess the hidden answer by interacting with the user in multiple turns.
 2. Goals: - You need to guess the answer to the puzzle within given interaction turns.
 3. User property: - The user hides the answer which you need to figure out.
 4. Interaction rules: - Rule 0: The user will first tell you the rule of the puzzle, and the interaction format that must be followed when querying. In each turn, the user will tell ***current turn*** and ***remaining turns***. - Rule 1: You can ask questions according to the rules of the game and receive corresponding feedback. If your ask is unavailable, the user will tell you. - Rule 2: After a series of interactions, you should answer the puzzle in the format specified in the description.
 5. Output format: - When you ask a question, you should strictly follow query format in the ****Description****. - When you answer the puzzle, you should strictly follow the answer format in the ****Description****. - Refer to the examples in the ****Description****. for the correct format. - If you figure out the right answer before given turns, keep interacting with the puzzle to make sure your answer is correct.
 6. Evaluation: - When the given number of interactions is reached, you need to give your answer of the puzzle. ****You MUST ONLY output the answer itself in the format mentioned in the description, DO NOT contain more text.****

Now Let's Solve the Puzzle algorithm.

****Description****: description.

2837

2838

2839

2840

2841

2842

2843

2844

2845

2846

2847

2848

2849

2850

2851

2852

2853

2854

2855

2856

2857

2858

2859

2860

2861

F.2.6 GAME STRATEGY INFERENCE (GSI)

1. Task overview: - The user plays the role of an opponent who takes a fixed strategy in a game. But you don't know what the strategy is. You need to guess the hidden strategy of your opponent by interacting (playing game) with him in multiple turns.
 2. Goals: - You have 1 final goal: You need to guess your opponent's strategy and try to maximize your score in the game. The score might depend on winning rate, or minimal cost, etc.
 3. User property: - The user hides his game strategy which you need to figure out to win the game.
 4. Interaction rules: - To finish the goal, you need to interact with the user. The interaction rules are as follows: - Interaction Rule 0: The user will first tell you the rule of the game, and the interaction format that must be followed when playing. In each turn, the user will tell ***current turn*** and ***remaining turns***. - Interaction Rule 1: You can take actions according to the rules of the game and receive corresponding feedback, such as current game states. If your action is unavailable, the user will tell you. - Interaction Rule 2: We will first play a few times of the game to familiarize you with the rules and the behavior of your opponent. In this phase, your actions will not be recorded, and your score does not matter. You can make use of this phase to explore the game and understand the opponent's strategy. - Interaction Rule 3: After the ***exploration phase***, you will enter the ***evaluation phase***. We will only play the game for 1 final time, and your actions will be recorded. Your score will be calculated based on your actions in this final game.
 5. Output format: - ****You must strictly obey the output format rules, DO NOT output any unrelated text!****
 6. Evaluation: - When the given number of interactions is reached, the game ends and we'll calculate your ****score****

Now Let's Play the Game {algorithm}, the Description Is that {description}.

2858

2859

2860

2861

G DETAILS OF BLACK-BOXES IN THE ORACLE BENCHMARK

We introduce the detailed implementation of black-boxes in each task. The definition of easy and hard black-boxes are based on two factors: human understanding of black-box difficulty and the

2862 black-box's inherent difficulty. The designers ranked the black-box difficulty of each task based on
 2863 a partial order relationship. They ultimately discussed and collectively determined the sets for easy
 2864 and hard tasks. In the beginning of each subsection, we detail the basis for the difficulty division.
 2865

2866 **G.1 CODE INTENT INFERENCE (CII)**
 2867

2868 In the context of CII tasks, easy black-boxes represent simple algorithms, while hard black-boxes en-
 2869 compass more complex algorithms and problem-specific programs. We instruct LLM to use mean-
 2870 ingless letters as variable names (e.g., a, b), so the variable name will not result in the leakage of
 2871 code intent.

2872 **(Easy 1) Quicksort Recursion** This code implements the quick sort algorithm using recursion to
 2873 sort an array of integers. Always choose the last number in the list as pivot.

2875 **(Easy 2) Most Frequent Char** Given a string, find the character that appears most frequently. If
 2876 there are multiple characters with the same frequency, return the one that appears first in the string.
 2877 The input string can contain letters, digits, and punctuation marks. The output should be a single
 2878 character that is the most frequent in the input string.
 2879

2880 **(Easy 3) KMP** The Knuth-Morris-Pratt (KMP) algorithm is used for substring search. It prepro-
 2881 cesses the pattern to create a longest prefix-suffix (LPS) array, which is then used to skip unnecessary
 2882 comparisons in the text. This allows for efficient searching of a pattern within a text string.
 2883

2884 **(Easy 4) High Precision Divide** Simulate high-precision division a/b (including floating-point
 2885 results), without using any floating-point numbers in the code. Keep to 4 decimal places.
 2886

2887 **(Easy 5) High Precision Add** This algorithm performs addition of very large integers. By simu-
 2888 lating the columnar addition, it adds digits from the least significant (rightmost) position, handling
 2889 carries.
 2890

2891 **(Easy 6) Fib Recursion** Give n , it returns the n -th number in the Fibonacci sequence using recur-
 2892 sion.
 2893

2894 **(Easy 7) Factorial Recursion** Given an integer n , return the factorial of n using recursion. The
 2895 factorial of n (denoted as $n!$) is the product of all positive integers less than or equal to n .
 2896

2897 **(Easy 8) Exgcd** The Extended Euclidean Algorithm computes the greatest common divisor (GCD)
 2898 of two integers and also finds the coefficients (x, y) such that $ax + by = \gcd(a, b)$.
 2899

2900 **(Easy 9) Coins for Fowls** You have m coins and you want to buy exactly n fowls. Each rooster
 2901 costs 5 coin, each hen costs 3 coins, and each chick costs 1 coin. This code solves how many
 2902 Roosters, Hens, and Chicks can you buy.
 2903

2904 **(Easy 10) Bubble Sort** Bubble sort is a sorting algorithm that repeatedly steps through the list,
 2905 compares adjacent elements and swaps them if they are in the wrong order. The pass through the list
 2906 is repeated until the list is sorted.
 2907

2908 **(Easy 11) Random Algebraic Operations** Perform basic algebraic operations on three input int
 2909 numbers a, b, c . First, calculate $d = a + b + c$. Then, calculate $e = a \times b - c$. Finally, calculate
 $f = (e + d - 10)/2$.
 2910

2911 **(Hard 1) Sieve of Eratosthenes** Implement the Sieve of Eratosthenes algorithm to find all prime
 2912 numbers up to a given limit.
 2913

2914 **(Hard 2) Mergesort Recursion** Merge sort is a divide-and-conquer algorithm that sorts an array
 2915 by recursively splitting it into halves, sorting each half, and then merging the sorted halves back
 together.
 2916

2916 **(Hard 3) Manacher** Use Manacher’s algorithm to find the longest palindromic substring in a
 2917 given string. The algorithm works by transforming the input string to handle even-length palin-
 2918 dromes and then expanding around potential centers to find the longest palindrome efficiently.
 2919

2920 **(Hard 4) Heapsort** Heap sort is a comparison-based sorting algorithm that uses a binary heap
 2921 data structure. It first builds a max heap from the input data, then repeatedly extracts the maximum
 2922 element from the heap and rebuilds the heap until all elements are sorted.
 2923

2924 **(Hard 5) Complex Random Algebraic Operations** Perform basic algebraic operations on four
 2925 input int numbers a, b, c, d . First, calculate $e = a \times b + c \times d$. If this value is greater than 50,
 2926 $e = e - 10$, else $e = e + 10$. Then, calculate $f = e^2 - a - b - c - d$. If f is odd, $f = (f - 1)/2$,
 2927 else $f = f/2$. Finally, calculate $g = f - a$.
 2928

2929 **(Hard 6) Arithmetic Slices** A sequence is defined as an arithmetic progression if it contains at
 2930 least three elements and the difference between any two consecutive elements is constant. Given an
 2931 integer array, return the number of all arithmetic subsequences within the array.
 2932

2933 G.2 CIRCUIT RULE INFERENCE (CRI)

2935 Black-boxes in CRI task represents a boolean circuit without circle, whose input wires are a fixed
 2936 number of 0/1 bits, and consisted of "AND"/"OR"/"NOT" gates. Easy black-boxes are typically
 2937 formatted in sequence structure and implement basic function (e.g., xor), while some hard black-
 2938 boxes are formatted in tree structure and implement advanced function (e.g., add).
 2939

2940 **(Easy 1) Swap** For input size $n = 9$, let the first 4 output gates are the last 4 input wires, and last
 2941 5 output gates are first 5 input wires. Notice that we use $b[i]=a[j]$ and $a[j]$ to implement copy.
 2942

2943 **(Easy 2) Random Small** For input size $n = 4$, we construct a circuit of 8 gates without nothing
 2944 restriction. The circuit is not for some well-known purpose.
 2945

2946 **(Easy 3) Consequence** For input size $n = 8$, we construct a circuit output whether there are two
 2947 consecutive input wires are both 1. Let input wires are $a[i], b[i] = a[i-1]$ and $a[i], s[i] = b[i]$ or $s[i-1]$.
 2948

2949 **(Easy 4) Xor Sequence** For input size $n = 8$, we construct a circuit calculate prefix XOR. Notice
 2950 that $a \text{ xor } b = (a \text{ and } (\text{not } b)) \text{ or } ((\text{not } a) \text{ and } b)$, $s[i] = s[i-1] \text{ xor } a[i]$.
 2951

2952 **(Easy 5) Palindrome** For input size $n = 8$, let $a[i]$ be the input wires, and we construct a circuit
 2953 check whether the input wires form a palindrome and $a[1..4] = a[5..8]$. $b[i] = a[i]$ and $a[n-i+1], c[i]$
 2954 $= a[i]$ or $a[n-i+1]$, $d[i] = b[i]$ or $(\text{not } c[i])$, $e[i] = a[i]$ and $a[i+n/2]$, $f[i] = a[i]$ or $a[i+n/2]$, $g[i] = e[i]$ or
 2955 $(\text{not } f[i])$, $s[i] = s[i-1]$ and $d[i]$ and $g[i]$.
 2956

2957 **(Hard 1) Random Big** For input size $n = 7$, we construct a circuit of 16 gates without nothing
 2958 restriction. The circuit is not for some well-known purpose.
 2959

2960 **(Hard 2) Greater** For input size $n = 6$, we construct a circuit to check whether the number of
 2961 input 1 wires is greater than or equal to the number of 0 wires. If number of 1 is greater or equal,
 2962 there must be three input wires are 1. Let the input be $a[1..6], c[1..20]$ be whether there are three 1
 2963 wires. For example, $c[1] = a[1]$ and $a[2]$ and $a[3], c[2] = a[1]$ and $a[2]$ and $a[4]$, ..., $c[20] = a[4]$ and
 2964 $a[5]$ and $a[6]$. The result is $c[1]$ or ... or $c[20]$.
 2965

2966 **(Hard 3) Path** For input size $n = 9$, it can be treated as a 4×4 adjacent matrix G of a 3 nodes
 2967 directed graph, where the diagnol is not important. We should construct a circuit that compute
 2968 whether there are paths between every pair of nodes. Because every possible path has length less
 2969 than 2, $G'[i][j] = \text{OR} (G[i][k] \text{ and } G[k][j])$. Then G' is the result. Notice that $G[i][i]=1$ no matter
 what the input is.

2970 **(Hard 4) Matrixmul** For input size $n = 9$, it can be treated as a 3×3 matrix M in field Z_2 .
 2971 Calculate the matrix square M^2 . Notice that $M^2[i][j] = \text{XOR}(M[i][k] \text{ and } M[k][j])$. $a \text{ xor } b = (a \text{ and } 2972 \text{ not } b) \text{ or } (b \text{ and } (\text{not } a))$.
 2973

2974 **(Hard 5) Count** For input size $n = 7$, we construct a circuit to count how many input wires are
 2975 1. Construct a subcircuit of add 0/1: $x'[0] = x[0] \text{ xor } \text{input}$, $y[0] = x[0] \text{ and } \text{input}$, $x'[1] = x[1] \text{ xor } 2976 \text{ y}[0]$, $y'[1] = x[1] \text{ and } y[0]$, $x'[2] = x[2] \text{ xor } y[1]$. Copy the subcircuit n times. Notice that $a \text{ xor } b =$
 2977 (not a and b) or (not b and a).
 2978

2979 **(Hard 6) Arbitrary** For input size $n = 7$ and an arbitrary function $f : Z_2^7 \rightarrow Z_2$. We can
 2980 construct a circuit to compute f . Let $f(x_1) = 1$, $f(x_2) = 1$, ... $f(x_k) = 1$. $f(x) = (x = x_1) \text{ or } (x =$
 2981 $x_2) \text{ or } \dots \text{ or } (x = x_k)$, $(x = y)$ can be construct by $((x[1] \text{ and } y[1]) \text{ or } \text{not}(x[1] \text{ and } y[1]))$ and $((x[2] \text{ and } y[2]) \text{ or } \text{not}(x[2] \text{ and } y[2]))$ and ... We choose x_1, x_2, \dots, x_8 without any particular pattern and
 2982 construct the circuit according to that.
 2983

2984 **(Hard 7) And Tree** For an input size of $n = 8$, we construct a circuit with $n - 1$ gates. The
 2985 circuit forms a tree structure, where each gate has two children and the output is the AND of its
 2986 two children. The last gate outputs the AND of all input wires. The second-to-last gate outputs the
 2987 AND of input wires with indices $[1, n/2]$, while the third-to-last gate outputs the AND of input wires
 2988 with indices $[n/2+1, n]$. The last gate outputs the AND of the results from the second-to-last and
 2989 third-to-last gates...
 2990

2991 **(Hard 8) Add** For input size $n = 10$, let the input wires be $a[i]$, we construct a circuit to com-
 2992 pute $a[2]a[3]a[4] \dots + a[1]$, which means add the first bit to the binary number of the last $n-1$ bits.
 2993 result[i]=a[n-i+1] xor b[i-1], b[i]=a[n-i+1] and b[i-1], b[0]=1.
 2994

2995 **(Hard 9) Consequence k** For input size $n = 10$, we construct a circuit output whether there are
 2996 two consecutive input wires are both 1 (we should treat the input wires as a circle, which means $x[8]$
 2997 $x[9] x[10] x[1]$ are also consecutive. Let $s[i] = x[i]$ and $x[\text{next}(i)]$, $s'[i] = s[i]$ and $s[\text{next}(\text{next}(i))]$,
 2998 where $\text{next}(i) = i \bmod 10 + 1$. $s'[i]$ checks whether there is consecutive 1 begins at i .
 2999

3000 **(Hard) Compare 10** For input size $n = 10$ ($x_1, \dots, x_5, y_1, \dots, y_5$), we construct a circuit for com-
 3001 pare x and y . That is, we compare $(x_1, y_1)(x_2, y_2) \dots (x_5, y_5)$ in order. $a > b$ can be construct by
 3002 (a and (not b)).
 3003

3004 G.3 PHYSICS SYSTEM INFERENCE (PSI)

3006 Easy black-boxes contain the most basic classical mechanics laws governing the fundamental motion
 3007 of one or two objects, while hard black-boxes contain two or more objects with more compli-
 3008 cated mechanics laws. Although the motion trajectories of most objects in the implemented black
 3009 box are one-dimensional or two-dimensional, for the sake of universality, we still performed three-
 3010 dimensional modeling for all object trajectories. In some cases (Double Pendulum, Harmonic with
 3011 Friction, Ball Air Resistance), equations of motion do not have analytical solutions. In these situa-
 3012 tions, we instruct LLMs to output code containing numerical solutions for the differential equations.
 3013 Then we compare black-box output with code output on test samples.
 3014

3015 **(Easy 1) Two Balls Collision Type 1** Two balls, A and B, are on an infinitely long, smooth plane.
 3016 Ball A has a mass of 5 kg, and Ball B has a mass of 2 kg. Ball A starts from the origin and moves
 3017 uniformly along the x-axis at a speed of 4 m/s. Ball B is placed at rest 10 m from the origin. When
 3018 the two balls collide, they bounce off each other completely elastically.
 3019

3020 **(Easy 2) Two Balls Collision Type 2** Two balls, A and B, are on an infinitely long, smooth plane.
 3021 Ball A has a mass of 4 kg, and Ball B has a mass of 2 kg. Ball A starts from the origin and moves
 3022 uniformly along the x-axis at a speed of 3 m/s. Ball B is placed at 12 m from the origin and come
 3023 to Ball A at a speed of 5 m/s. When the two balls collide, they bounce off each other completely
 elastically.

3024 **(Easy 3) Simple Harmonic Type 1** A simple harmonic oscillator is a system that experiences a
 3025 restoring force proportional to the displacement from its equilibrium position. The system contains
 3026 one object. The mass of the object is 1 kg. The spring constant is 100 N/m. The initial displacement
 3027 from the equilibrium position is 0.2 m (spring in contracted state). The system is horizontal and the
 3028 object is support by smooth ground without friction. The system is released from rest and do not
 3029 consider any air resistance. The coordinate origin is set at the equilibrium position of the oscillator.
 3030

3031 **(Easy 4) Simple Harmonic Type 2** A simple harmonic oscillator is a system that experiences a
 3032 restoring force proportional to the displacement from its equilibrium position. The system contains
 3033 one object. The mass of the object is 1 kg. The spring constant is 100 N/m. The initial displacement
 3034 from the equilibrium position is 0.2 m (spring in contracted state). The system is vertical and the
 3035 gravity is 10 m/s^2 . The system is released from rest and do not consider any air resistance. The
 3036 coordinate origin is set at the initial position of the oscillator.

3037 **(Easy 5) Oblique Projectile Motion** An oblique projectile motion system is a scenario where an
 3038 object is projected at an angle from a very high point and falls under the influence of gravity. The
 3039 system contains one object. The mass of the object is 2 kg. The initial height from which the object
 3040 is projected is infinite, meaning it starts at a very high altitude. The initial horizontal velocity of
 3041 the object is 10 m/s, and the initial vertical velocity is 10 m/s. The simulation does not consider air
 3042 resistance. The coordinate origin is set at the initial position. Gravity g is 10 m/s^2 .
 3043

3044 **(Easy 6) Pendulum** A pendulum is a weight suspended from a pivot so that it can swing freely.
 3045 The system contains one object. The initial angle of the pendulum is 60 degree. The length of the
 3046 pendulum is 2 meters. The acceleration due to gravity is 10 m/s^2 . Do not take air resistance into
 3047 account. The pendulum is released from rest. The coordinate origin is set at the pivot point of the
 3048 pendulum.

3049 **(Easy 7) Elliptical Planetary Orbits** An elliptical planetary orbit is a path followed by a planet
 3050 around a star, where the orbit is an ellipse. The system contains one object which is the planet. The
 3051 mass of the planet is $9 \times 10^{24} \text{ kg}$ and the mass of the star is $8 \times 10^{29} \text{ kg}$. The periapsis is 6×10^{10}
 3052 m and the apoapsis is 9×10^{10} m. The simulation does not consider any other forces except for the
 3053 gravitational force between the planet and the star. The coordinate origin is set at the center of the
 3054 star. G equals to $6.7 \times 10^{-11} \text{ Nm}^2/\text{kg}^2$.
 3055

3056 **(Easy 8) Cycloid** A cycloid is the curve traced by a fixed point on the circumference of a circle
 3057 as it rolls along a straight line without slipping. The system contains one object which is the fixed
 3058 point. The radius of the circle is 1 m. The circle moves in a uniform speed of 1 m/s. The coordinate
 3059 origin is set at the fixed point where the circle starts rolling.

3060 **(Easy 9) Conical Pendulum** A conical pendulum is a pendulum that moves in a circular path
 3061 while swinging at an angle. The system contains one object which is the pendulum bob. The mass
 3062 of the pendulum bob is 2 kg. The length of the pendulum is 5 m. The angle between the string and
 3063 the vertical is 30 degrees. The simulation does not consider air resistance. The coordinate origin is
 3064 set at the point where the string is attached to the ceiling. Gravity g is 10 m/s^2 .
 3065

3066 **(Easy 10) Free Fall to the Ground Type 1** A free fall system is a scenario where an object
 3067 falls under the influence of gravity until it collides with the ground, and then bounces. The system
 3068 contains one object. The mass of the object is 5 kg. The initial height from which the object is
 3069 dropped is 10 m. The ground is modeled as a surface that can completely deform elastically upon
 3070 impact, meaning it will return to its original shape after the collision. The simulation does not
 3071 consider air resistance. The coordinate origin is set at the ground level. Gravity g is 10 m/s^2 .
 3072

3073 **(Easy 11) Free Fall Infinite Height** An object is dropped from an infinite height. The acceleration
 3074 due to gravity is 10 m/s^2 . The object falls freely under the influence of gravity. Do not take air
 3075 resistance into account. The coordinate origin is set at the point where the object is dropped.

3076 **(Easy 12) Horizontal Projectile Motion** A horizontal projectile motion system is a scenario
 3077 where an object is projected horizontally from a very high point and falls under the influence of

3078 gravity. The system contains one object. The mass of the object is 2 kg. The initial height from
 3079 which the object is projected is infinite, meaning it starts at a very high altitude. The initial horizontal
 3080 velocity of the object is 10 m/s. The simulation does not consider air resistance. The coordinate
 3081 origin is set at the initial position. Gravity g is 10 m/s².
 3082

3083 **(Easy 13) Free Fall to the Ground Type 2** A free fall system is a scenario where an object
 3084 falls under the influence of gravity until it collides with the ground, and then bounces. The system
 3085 contains one object. The mass of the object is 5 kg. The initial height from which the object is
 3086 dropped is 20 m. The ground is modeled as a surface that deform inelastically upon impact. The
 3087 coefficient of restitution is 0.6. The simulation does not consider air resistance. The coordinate
 3088 origin is set at the ground level. Gravity g is 10 m/s².
 3089

3090 **(Hard 1) Boat in River** A rectangular river is 2000 m long and 20 m wide. The river's flow
 3091 velocity is proportional to the distance from the river banks, with the velocity being 0 at both banks
 3092 and 10 m/s at the center of the river. A boat travels from one bank to the other perpendicular to the
 3093 current at a constant relative speed of 2 m/s. At a point 5 m from the bank, due to a malfunction,
 3094 the boat immediately turns around and heads back towards the original bank perpendicular to the
 3095 current at a speed of 1 m/s.
 3096

3097 **(Hard 2) Block Released from Incline Plane** An inclined plane with an angle of 30 degrees and
 3098 a mass of 10 kg is placed on a smooth horizontal surface. A wooden block with a mass of 5 kg
 3099 is placed at a perpendicular height of 8 m on the inclined plane. There is no friction between the
 3100 wooden block and the inclined plane. In the initial state, the wooden block's coordinates are (0, 0,
 3101 8) and the inclined plane's coordinates are (0, 0, 0). The system is released from rest. After the
 3102 wooden block separates from the inclined plane, both the wooden block and the inclined plane will
 3103 move along the smooth horizontal surface. $g = 10$ m/s².
 3104

3105 **(Hard 3) Three Balls Collision** There are three balls, A, B, and C, with masses of 4 kg, 3 kg, and
 3106 2 kg, respectively. Their initial positions are at 0 m, 10 m, and 20 m, and their initial velocities are 3
 3107 m/s, 3 m/s, and -4 m/s. The plane is 30 m long with walls at both ends. The collisions between the
 3108 balls and between the balls and the walls are all perfectly elastic. The plane is perfectly smooth.
 3109

3110 **(Hard 4) Harmonic with Friction** A harmonic oscillator with friction is a system that exper-
 3111 iences a restoring force proportional to the displacement from its equilibrium position, along with a
 3112 damping force proportional to the velocity. The system contains one object. The mass of the object
 3113 is 1 kg. The spring constant is 100 N/m. The initial displacement from the equilibrium position is 1
 3114 m (spring in contracted state). The system is horizontal and the object is supported by ground with
 3115 dynamic friction coefficient 0.1, gravity equals to 10. The system is released from rest and do not
 3116 consider any air resistance. The coordinate origin is set at the equilibrium position of the oscillator.
 3117 The simulation time begins from 0 s to 100 s with a time step of 0.1 s.
 3118

3119 **(Hard 5) Double Pendulum** A double pendulum is a mechanical system that consists of two
 3120 pendulums attached end to end. The first pendulum is attached to a fixed pivot, and the second
 3121 pendulum is attached to the free end of the first pendulum. The system contains two objects. The
 3122 first object weighs 1 kg and attached to a fixed pivot with a 1 meter rigid rod. The first object weighs
 3123 1 kg and attached to the first object with a 1 meter rigid rod. The initial angle of the first pendulum
 3124 is 45 degrees, and the initial angle of the second pendulum is 45 degrees. The acceleration due to
 3125 gravity is 10 m/s². Do not take air resistance into account. The coordinate origin is set at the pivot
 3126 point of the first pendulum. The simulation time begins from 0 s to 100 s with a time step of 0.1 s.
 3127 The system is released from rest.
 3128

3129 **(Hard 6) Ball Air Resistance** A 2 kg ball is thrown vertically upward from a horizontal surface
 3130 with an initial velocity of 15 m/s. The air resistance is given by $f=0.1v^2$, where v is the ball's
 3131 velocity. After reaching its highest point, the ball keeps falling as there's no ground. $g = 10$ m/s².
 3132

3133 **(Hard 7) Two Balls Collision Type 3** Two balls, A and B, are on a 20 m, smooth plane with walls
 3134 on each side. Ball A has a mass of 5 kg, and Ball B has a mass of 3 kg. Ball A starts from one side
 3135 of the plane and moves uniformly along the plane at a speed of 4 m/s. Ball B starts from the other
 3136

3132 side of the plane and moves towards Ball A at a speed of 6 m/s. When the two balls collide, the
 3133 collision is inelastic and the coefficient of restitution $e = 0.8$. When the balls collide with the walls,
 3134 they bounce off elastically.
 3135

3136 **(Hard 8) Bullet** A 2000 g object, 60m in length, slides on a horizontal surface with a coefficient
 3137 of friction of 0.1. Its initial speed is 15 m/s. A 50 g bullet is flying horizontally at 400 m/s. After
 3138 1.5 seconds, the bullet horizontally embeds into the object. During the bullet's passage through the
 3139 object, it experiences a resistive force $f = 0.2v$, where v is the bullet's instantaneous velocity. When
 3140 the bullet goes through the object, it continues horizontal uniform flight. Neglect the bullet's vertical
 3141 drop. Use $g = 10 \text{ m/s}^2$.
 3142

3143 G.4 ENCRYPTION RULE INFERENCE (ERI)

3144 For every letter, its plaintext and ciphertext correspond in all cases for most of the easy black-boxes.
 3145 Some easy black-boxes (e.g. Zigzag Cipher and Curve Cipher) change the order of plaintext via spe-
 3146 cific algorithm. For hard black-boxes, the encrypted letters barely retain their original frequencies,
 3147 which adds difficulty in decryption.
 3148

3149 **(Easy 1) 01248 Encryption Type 2** For characters A-Z or a-z (case-insensitive), each letter is
 3150 assigned a value from 1 to 26 (a/A=1, b/B=2, etc.). This value is then broken down into the sum of
 3151 combinations of 1, 2, 4, 8. Use larger numbers first to sum. Then calculate the amount of 1, 2, 4,
 3152 8 separately and convert it to a four-digits number where the ones, tens, hundreds, and thousands
 3153 places represent the amounts of 1, 2, 4, 8, respectively. For example, C=1+2=0011, j=2+8=1010,
 3154 Z=2+8+8+8=3010. Black space is replaced by 0.
 3155

3156 **(Easy 2) Backpack Encryption** It uses a superincreasing sequence as private key. The plaintext
 3157 is a string of letters. The encryption process is as follows: Convert each letter to a binary value:
 3158 $a/A=00000, b/B=00001, \dots, z/Z=11001$. The public key is set as $\{34, 51, 58, 11, 35\}$. The ciphertext
 3159 is obtained by summing the products of each public key element and its corresponding digit in the
 3160 binary representation. For example, the letter 'c' is encrypted to $(0 \times 34) + (0 \times 51) + (0 \times 58) +$
 3161 $(1 \times 11) + (0 \times 35) = 11$. Output the ciphertext as a string of numbers, separated by commas.
 3162

3163 **(Easy 3) Caesar Cipher** Caesar cipher is a type of substitution cipher in which each letter in the
 3164 plaintext is shifted 8 places down the alphabet. For example, A(a) would be replaced by I(i), B(b)
 3165 would become J(j), and so on. For letters at the end of the alphabet, the shift wraps around to the
 3166 beginning. For example, Y(y) would become G(g). Only take letters A-Z and a-z (case-sensitive)
 3167 into account. Blank spaces are ignored.
 3168

3169 **(Easy 4) ADHXZ Encryption** ADHXZ is a method of encryption that uses a 5×5 grid to encrypt.
 3170 The encryption is case-insensitive and follows this replacement rule: 'a/A=AH, b/B=AA, c/C=XH,
 3171 d/D=DA, e/E=ZH, f/F=HD, g/G=XA, h/H=DD, i/I=XD, j/J=XD, k/K=DZ, l/L=AX, m/M=ZA,
 3172 n/N=HZ, o/O=DH, p/P=AZ, q/Q=HA, r/R=ZD, s/S=HX, t/T=AD, u/U=XX, v/V=HH, w/W=ZX,
 3173 x/X=XZ, y/Y=ZZ, z/Z=DX'. Black spaces are kept.
 3174

	A	D	H	X	Z
A	b/B	t/T	a/A	i/L	p/P
D	d/D	h/H	o/O	z/Z	k/K
H	q/Q	f/F	v/V	s/S	n/N
X	g/G	i/I/j/J	c/C	u/U	x/X
Z	m/M	r/R	e/E	w/W	y/Y

3180 **(Easy 5) Bacon Cipher** This variant of Bacon's cipher is a method of encoding text using a binary
 3181 system. Each letter of the alphabet is represented by a unique combination of five letters, either
 3182 'X'('x') or 'Y'('y'). 'X'('x') indicates 0 and 'Y'('y') indicates 1. The encryption is case-sensitive,
 3183 meaning uppercase letters will be replaced by 'X', 'Y', and lowercase letters will be replaced by
 3184 'x', 'y'. The first thirteen letters of the alphabet, namely 'A' to 'M' will be replaced by 'XXXXXX'
 3185 to 'YYYYXX', while the last thirteen letters, namely 'N' to 'Z' will be replaced by 'YXXXYY' to
 'YYYYYY'. Blank spaces are ignored.
 3186

3186 **(Easy 6) Zigzag Cipher** Write the letters of plaintext in a zigzag path, moving downwards and
 3187 then upwards across the 3 rows. Imagine writing on 3 parallel lines: when you reach the bottom
 3188 row, move back up, and when you reach the top row, move back down, repeating this pattern until
 3189 all plaintext characters are written. Spaces are ignored. After writing, read the characters from left
 3190 to right, row by row, to form the ciphertext. For example, the zigzag path for ‘HELLO WORLD’ is
 3191

H		O		L	
E	L	W	R	D	
L		O			

3195 So it’s encrypted as ‘HOLELWRDLO’.
 3196

3197 **(Easy 7) Fibonacci Encryption** Fibonacci cipher is a method of encryption that uses the Fibonacci
 3198 sequence to determine the shift of each letter in the plaintext. Suppose each letter is represented by
 3199 $a=0, b=1, \dots, z=25, A=26, B=27, \dots, Z=51$. The value of each encrypted letter is its value in plaintext
 3200 plus a shift number from the Fibonacci sequence and then modulo 52. The Fibonacci sequence
 3201 starts with 1 and 1, and each subsequent number is the sum of the two preceding ones. The first few
 3202 numbers in the sequence are 1, 1, 2, 3, 5, 8, 13, ... For example, if the plaintext is ‘HELLO’, the
 3203 first letter ‘H’ is shifted by 1 (the first Fibonacci number), resulting in ‘I’. The second letter ‘E’ is
 3204 shifted by 1 (the second Fibonacci number), resulting in ‘F’. The third letter ‘L’ is shifted by 2 (the
 3205 third Fibonacci number), resulting in ‘N’. The fourth letter ‘L’ is shifted by 3 (the fourth Fibonacci
 3206 number), resulting in ‘O’. The fifth letter ‘O’ is shifted by 5 (the fifth Fibonacci number), resulting
 3207 in ‘T’. Therefore, the ciphertext is ‘IFNOT’. Keep the blank space in the ciphertext.
 3208

3209 **(Easy 8) Index Shift Encryption** Index shift encryption is a method of encryption that uses the
 3210 index of each letter in the plaintext to determine the shift for each letter. The index of first letter is
 3211 0. Suppose the letters are represented by $a=0, b=1, \dots, z=25, A=26, B=27, \dots, Z=51$. The value of
 3212 each letter in ciphertext equals its value in plaintext plus a shift number (i.e. the number of index)
 3213 and then modulo 52. Then convert the value to the corresponding letter. Keep the blank spaces in
 3214 the ciphertext.
 3215

3216 **(Easy 9) Curve Cipher** There is a table with 999 rows and 6 columns. Each letter in plaintext is
 3217 sequentially filled into the table row by row. Blank spaces are ignored. Uppercase and lowercase
 3218 are kept. When all the plaintext letters are filled into the table, the ciphertext is generated by reading
 3219 letters in the table in a zigzag manner, beginning from the last letter in the last available column
 3220 (e.g. begin from ‘t’ in the 2-th column for ‘at’ and begin from ‘i’ in the 6-th column for ‘beautiful’).
 3221 When reaching the top row, move back down, and when reaching the bottom row, move back up,
 3222 repeating this pattern until all the letters are read. Letters in the same column are written together
 3223 and letters in different columns are separated by blank space. For example, if the plaintext is ‘Hello
 3224 World’, the table will be like
 3225

H	e	l	l	o	W
o	r	1	d		

3226 And the ciphertext is supposed to be ‘W o dl ll re Ho’.
 3227

3228 **(Easy 10) 01248 Encryption Type 1** For characters A-Z or a-z (case-insensitive), the encryption
 3229 is done by first replacing each character with a number based on its position in the alphabet, namely,
 3230 $a/A=1, b/B=2, \dots, z/Z=26$. Each number in 1 to 26 is replaced by the sum of combinations of 1, 2,
 3231 4, 8. Black space is replaced by 0. For example, 3 is replaced by 12, 10 is replaced by 28, 20 is
 3232 replaced by 488. The replacement follows two rule: (1). Use larger numbers first to make sure the
 3233 encrypted number has the fewest digits (e.g. use 18 for 9 instead of 1224). (2). Put small number in
 3234 the front. (e.g. use 18 instead of 81). According to the above rules, string ‘Hello World’ is encrypted
 3235 to ‘814484812480124881248288484’.
 3236

3237 **(Easy 11) Substitution Encryption** Simple substitution cipher is a method of encryption where
 3238 each letter in the plaintext is replaced by a letter with a fixed relationship to it. The detailed corre-
 3239 sponding relationship is as follows: Each letter in ‘abcdefghijklmnopqrstuvwxyz’ will be replaced
 3240 by ‘phqgiumeaylnofdxjkrcvstzwb’. The process is case-sensitive, meaning uppercase will keep as

3240 uppercase and lowercase will keep as lowercase. If the plaintext contains blank spaces, keep them
 3241 in the ciphertext.
 3242

3243 **(Hard 1) Sequential Feedback Cipher** Sequential feedback cipher encrypts letter in the plaintext
 3244 one by one in a sequential order. Each letter in the alphabet is assigned a value A=0, B=1, ..., Z=25,
 3245 a=26, b=27, ..., z=51. The initial hidden letter is set as ‘b’. For each letter in the plaintext, the
 3246 shift number is the value of previous encrypted letter. And for the first letter, the shift number is the
 3247 value of initial hidden letter. The ciphertext for each letter is its original value plus the shift number
 3248 and then modulo 52. For example, the letter ‘A’ in plaintext ‘At’ will be encrypted to ‘b’ because
 3249 $(27 + 0) \bmod 52 = 27$. The letter ‘t’ will be encrypted to ‘U’ because $(45 + 27) \bmod 52 = 20$.
 3250 Blank spaces are kept in the ciphertext.

3251
 3252 **(Hard 2) RSA Encryption** The public key for RSA is set as $(e, n) = (13, 713)$, and the private
 3253 key is set as $(d, n) = (1117, 713)$. The plaintext is a string of letters. The encryption process is as
 3254 follows: First, convert each letter to its corresponding integer value: a/A=1, b/B=2, ..., z/Z=26. This
 3255 procedure is case-insensitive. Then, split the plaintext into blocks with a size of 2. Calculate the
 3256 value of each block in hexavigesimal. For example, the value of ‘HE’ is $8 \times 26 + 5 = 213$. Finally,
 3257 perform traditional RSA algorithm to each block. Blank spaces in plaintext are ignored. Output the
 3258 ciphertext as a string of integers, separated by commas.

3259
 3260 **(Hard 3) Frequency Encryption** For each letter in the plaintext, its ciphertext is affected by
 3261 the frequency of letters before it in the plaintext. Suppose we have plaintext $P = p_1p_2...p_n$ and
 3262 the initial hidden keyword p_0 is set as ‘H’. A-Z and a-z are represented by numerical values 0-
 3263 25 and 26-51 respectively. For letters $p_i (i \geq 1)$, the value for encrypted letter c_i is calculated as
 3264 $c_i = (p_i + \text{shift}_i) \bmod 52$, where shift_i is the value of the letter that has the highest frequency in
 3265 $p_0p_1...p_{i-1}$. If several letters have the same frequency, choose the letter according to the following
 3266 preference order: ‘A’>‘B’>...>‘Z’>‘a’>‘b’>...>‘z’. Keep the blank spaces in the ciphertext.

3267
 3268 **(Hard 4) Dynamic Curve Cipher** There is a table with 999 rows and k columns, where $k =$
 3269 $(l \bmod 3) + 3$, and l is the number of letters in the plaintext. Each letter in plaintext is sequentially
 3270 filled into the table row by row. Blank spaces are ignored. Uppercase and lowercase are kept. When
 3271 all the plaintext letters are filled into the table, the ciphertext is generated by reading letters in the
 3272 table in a zigzag manner, beginning from the last letter in the last available column (e.g. begin from
 3273 ‘t’ in the 2-th column for ‘at’ and begin from ‘l’ in the 3-th column for ‘beautiful’). When reaching
 3274 the top row, move back down, and when reaching the bottom row, move back up, repeating this
 3275 pattern until all the letters are read. Letters in the same column are written together and letters in
 3276 different columns are separated by blank space. For example, if the plaintext is ‘Hello World’, the
 3277 ciphertext is supposed to be ‘rl lo dWe Hol’.

3278
 3279 **(Hard 5) Vigenere Encryption** Vigenere cipher is a method of encryption that uses a keyword
 3280 ‘MEMORY’ to encrypt the plaintext. Each letter in the plaintext is shifted by a number of positions
 3281 determined by the corresponding letter in the keyword. The keyword is repeated to match the length
 3282 of the plaintext. For example, if the plaintext is ‘HELLOWORLD’, then the keyword is repeated
 3283 as ‘MEMORYMEMO’. The first letter ‘H’ is shifted by ‘M’ (12 positions), resulting in ‘T’. The
 3284 second letter ‘E’ is shifted by ‘E’ (4 positions), resulting in ‘I’. The encryption is case-insensitive.
 3285 Both uppercase and lowercase letters will be treated as uppercase. Keep the blank space in the
 3286 ciphertext.

3287
 3288 **(Hard 6) Hill Cipher** The hill cipher is a polygraphic substitution cipher based on linear algebra.
 3289 Each letter is first mapped into a number (a/A=0, b/B=1, ..., z/Z=25). The hill cipher uses matrix
 3290 multiplication to encrypt blocks of text. The key is a 2×2 square matrix $K = \begin{pmatrix} 3 & 5 \\ 1 & 2 \end{pmatrix}$, and the
 3291 plaintext is divided into blocks with a length of 2. If the last block is fewer than 2 letters, use ‘x’
 3292 to fill. Each block is represented as a 2-D vector, and the encryption is performed by multiplying
 3293 the key matrix with the plaintext vector modulo 26. The resulting vector is then converted back to
 letters. The encryption process is case-insensitive. Output the ciphertext in lowercase style. For
 example, ‘Hi’ will be encrypted to ‘jx’. Keep the blank spaces in the ciphertext.

3294 **(Hard 7) Positional Keyword Cipher** This method uses a keyword ‘Jackal’ to determine how
 3295 each letter in the plaintext is shifted. Each letter of the alphabet is assigned a value A=0, B=1, ...,
 3296 Z=25, a=26, b=27, ..., z=51. Write the keyword repeatedly above the plaintext, matching each letter
 3297 of the plaintext with a letter from the keyword. Blank spaces are kept. For example, if plaintext
 3298 is ‘Hello World’, then the corresponding keyword for matching is ‘Jacka lJack’. For each letter,
 3299 calculate the sum of the value of plaintext and keyword, then modulo 52 to get the value of the ci-
 3300 phertext letter. Finally, translate the value to corresponding letter. Since the keyword is repeated, the
 3301 same letter in the plaintext will be paired with different letters of the keyword at different positions,
 3302 leading to different ciphertext letters.

3303 **(Hard 8) Compositional Encryption** It’s an encryption method that undergoes polyalphabetic
 3304 substitution, affine transformation, and fixed permutation. The encryption process follows: For each
 3305 English letter P_i in the plaintext: (1). Letter to Number Conversion: Convert P_i to its corresponding
 3306 number p_i (a/A=0, ..., z/Z=25). (2). Get Keyword Character: Let the keyword be K with length
 3307 L_k . Take the $j = (i \bmod L_k)$ -th character K_j from the keyword and convert it to its numerical
 3308 value k_j . The keyword K is set as ‘LOVE’. (3). Polyalphabetic Substitution: Calculate $s_{1,i} =$
 3309 $(p_i + k_j) \bmod 26$. (4). Affine Transformation: Calculate $s_{2,i} = (3 \times s_{1,i} + 10) \bmod 26$. (5). Fixed
 3310 Permutation: Generate a permutation table P_{table} and use it to find $s_{3,i} = P_{table}[s_{2,i}]$. (6). Number
 3311 to Letter Conversion: Convert $s_{3,i}$ back to its corresponding letter C_i . (7). Keep Blank Spaces:
 3312 Keep the blank spaces in the ciphertext.

3313 **(Hard 9) Playfair Cipher** Playfair is a method of encryption. It encrypts pairs of letters using
 3314 a 5×5 grid of letters constructed from a keyword ‘SECURITY’. The keyword is used to fill the
 3315 grid, and the remaining letters of the alphabet are filled in order, skipping any letters already in the
 3316 keyword. The letter ‘J’ is combined with ‘I’. To encrypt, locate each letter pair in the grid: If they
 3317 are in the same row, replace them with letters to their right; If they are in the same column, replace
 3318 them with letters below; If they form a rectangle, replace them with letters on the same row but at
 3319 the opposite corners. If a pair consists of two identical letters, insert a filler letter ‘X’ between them.
 3320 If the plaintext has an odd number of letters, append a filler letter ‘X’ at the end. The ciphertext is
 3321 case-sensitive and blank spaces are considered.

3323 G.5 INTERACTIVE PUZZLE INFERENCE (IPI)

3324 Easy black-boxes in IPI task are puzzles with simple rules and limited action space, while hard
 3325 black-boxes have more complex rules and a much larger range of actions to choose from.

3326 **(Easy 1) Three Arms Bandit** In the 3-Arm Bandit problem, there are 3 slot machines named
 3327 ‘Bandit A’, ‘Bandit B’, and ‘Bandit C’. Each machine has a probability of winning, which is un-
 3328 known to you. The player’s goal is to discover the better bandit by choosing which machine to play
 3329 over a series of rounds. You can choose either ‘Bandit A’, ‘Bandit B’, or ‘Bandit C’ in each round,
 3330 and you will receive a reward (such as ‘Reward: 0’ or ‘Reward: 1’) based on the machine’s winning
 3331 probability. After a series of rounds, player need to answer ‘Bandit A’, ‘Bandit B’ or ‘Bandit C’ to
 3332 indicate which bandit they believe has the higher winning probability.

3333 **(Easy 2) Single Battleship** Single Battleship is a simplified version of the classic Battleship game.
 3334 In this game, the player has to guess the location of a single 1×5 battleship on a 9×9 grid. The player
 3335 will make guesses by specifying coordinates on the grid, in the form of ‘(x, y)’, from ‘(1, 1)’ to ‘(9,
 3336 9)’, and the game will provide feedback on whether the guess ‘hits’ or ‘misses’ the battleship. After
 3337 a series of guesses, the player will provide the coordinates of the battleship in the format ‘Row X’
 3338 or ‘Column Y’, where ‘X’ and ‘Y’ are integers from 1 to 9, since a battleship must be placed either
 3339 horizontally or vertically and occupies a whole row or column.

3340 **(Easy 3) Wordle** Wordle is a word-guessing game where players attempt to deduce a hidden 8-
 3341 letter word (all uppercase). Each guess provides feedback (1).Correct letter in the correct position,
 3342 represented by ‘A’. (2).Correct letter but misplaced, represented by ‘M’. (3). Letter not in the word,
 3343 represented by ‘X’. Players iteratively refine guesses using feedback until solving the word, typically
 3344 within limited attempts. After a few times of querying, the player will give an 8-letter uppercase
 3345 word answer, and he’ll win if he answers correctly. Output the 8-letter uppercase word directly.

3348 **(Easy 4) Heavy Coin** In the Heavy Coin puzzle, there are 100 coins, named ‘Coin 1’, ‘Coin 2’,
 3349 ..., ‘Coin 100’. Among these coins, one is heavier than the others. The task is to identify the heavy
 3350 coin using a balance scale. The balance scale has two properties: (1).It can compare the weight of
 3351 two groups of coins at a time. But it only returns balance or imbalance, without giving which side
 3352 is heavier. (2).It will lie once in a random turn in every 10 interactions. When it lies, it will return
 3353 the opposite of the truth. Player must make queries with the same format of this example: ‘Left:
 3354 Coin 1, Coin 2, Coin 3; Right: Coin 4, Coin 5, Coin 6’. The balance scale will return ‘Balance’ if
 3355 both sides are balanced or ‘Imbalance’ if one side is heavier. After a series of queries, the player
 3356 need to identify the heavy coin among the 100 coins in the form of ‘Heavy Coin X’, where ‘X’ is
 3357 the number of the heavy coin.

3358 **(Easy 5) Number Guessing** In the Number Guessing game, the player will try to guess a secret
 3359 integer that is randomly chosen from $[0, 100]$. The player will make guesses in the form of ‘Number
 3360 X’ (such as ‘Number 10’). And after each guess, the player will receive feedback indicating whether
 3361 the absolute value of guess minus ground truth number is ‘Greater than 15’ or ‘Less than or equal to
 3362 15’. If the absolute value is ‘Greater than 15’, the player will receive ‘Far’, else if the absolute value
 3363 is ‘Less than or equal to 15’, the player will receive ‘Close’. After a series of guesses, the player
 3364 needs to answer with the correct number they believe is the secret integer in the form of ‘Number
 3365 X’.

3366 **(Hard 1) Heavy Light Coins** In the Heavy Light Coins puzzle, there are 152 coins, named ‘Coin
 3367 1’, ‘Coin 2’, ..., ‘Coin 152’. Among these coins, one is heavier than the others, and another one is
 3368 lighter than the others. If the normal coin weighs 1, then the heavy coin weighs 1.1 and the light
 3369 coin weighs 0.9. The task is to identify the heavy coin and the light coin using a balance scale. The
 3370 balance scale can compare the weight of two groups of coins at a time. For example, player can
 3371 make queries like ‘Left: Coin 1, Coin 2, Coin 3; Right: Coin 4, Coin 5, Coin 6’. The balance scale
 3372 will return ‘Left’ if the left side is heavier, ‘Right’ if the right side is heavier, or ‘Equal’ if both sides
 3373 are balanced. After a series of queries, the player need to identify the heavy coin and the light coin
 3374 among the 152 coins in the form of ‘Heavy Coin X; Light Coin Y’, where ‘X’ and ‘Y’ is the number
 3375 of the heavy coin and the light coin.

3376 **(Hard 2) Wordle Hard** Wordle is a word-guessing game where players attempt to deduce a hid-
 3377 den 11-letter word (all uppercase). Each guess provides feedback (1).Correct letter in the correct
 3378 position, represented by ‘A’. (2).Correct letter but misplaced, represented by ‘M’. (3).Letter not in
 3379 the word, represented by ‘X’. Players iteratively refine guesses using feedback until solving the
 3380 word, typically within limited attempts. After a few times of querying, the player will give an 11-
 3381 letter uppercase word answer, and he’ll win if he answers the correct word. Output the 11-letter
 3382 uppercase word directly.

3383 **(Hard 3) Nerdle** Nerdle is a math-based puzzle game where players guess a hidden equation
 3384 consisting of numbers and operators. Here we guarantee that the length of the equation is 15,
 3385 including digits (0-9) and operators (+, -, *, /, =). For example, truth can be ‘10002*3+6=30012’.
 3386 In each turn, player can guess a length 15 valid equation (such as ‘1+1+1+1=50’ is invalid since this
 3387 is not correct, and ‘1+1+1+1=4’ is invalid since length is only 9). Each guess provides feedback:
 3388 (1) Correct digit/operator in the correct position, represented by ‘A’; (2) Correct digit/operator but
 3389 misplaced, represented by ‘M’; (3) Digit/operator not in the equation, represented by ‘X’. Players
 3390 iteratively refine guesses using feedback until solving the equation, within limited attempts. After a
 3391 few turns of querying, the player will give a 15-character valid equation answer.

3392 **(Hard 4) Nuts and Bolts** There are 8 bolts and 8 nuts of different sizes, named ‘Bolt 1’, ..., ‘Bolt’
 3393 and ‘Nut 1’ ... ‘Bolt 8’, where each bolt exactly matches one nut. We know that ‘Bolt 1’ is smaller
 3394 than ‘Bolt 2’, ‘Bolt 2’ is smaller than ‘Bolt 3’, and so on. The goal is to find the matching nut for
 3395 each bolt. In each turn, the player can make a query of 3 bolt-but pairs in the form of ‘Bolt X1,
 3396 Nut Y1; Bolt X2, Nut Y2; Bolt X3, Nut Y3’, which returns ‘small’, ‘large’ or ‘equal’ to represent
 3397 whether the nut is too small, too large, or a match for the bolt, for example ‘small; small; large’.
 3398 After a series of queries, the player must answer a permutation of the nuts, such as ‘Nut 1, Nut 3,
 3399 Nut 2, Nut 5, Nut 4, Nut 6, Nut 7, Nut 8’, which represents the order of nuts that match the bolts

3402 from ‘Bolt 1’ to ‘Bolt 8’. The player will win if they correctly identify the matching nuts for all
 3403 bolts.

3404

3405 **(Hard 5) Battleship** Battleship is a strategic guessing game where the player attempts to figure
 3406 out the location of 3 hidden ships on a 7×7 grid. 3 ships are 1×3 , 1×4 and 1×5 respectively, and we
 3407 use ‘AAA’, ‘BBBB’ and ‘CCCC’ to represent them. The player will make guesses by specifying
 3408 coordinates on the grid, in the form of (x, y) , from $(1, 1)$ to $(7, 7)$, and the game will provide
 3409 feedback on whether the guess ‘hits’ or ‘misses’ any ships. Besides the ships, there are two 1×1
 3410 bombs denoted as ‘O’. When the coordinate is bomb, the game will also provide ‘hits’. Note that
 3411 the player can only know whether a ship or a bomb is attacked, but not which ship or bomb it is.
 3412 After a series of guesses, the player need to distinguish the bombs and provide the coordinates of the
 3413 ships by drawing a grid with ‘A’, ‘B’ and ‘C’ representing the ships, and ‘.’ representing empty water
 3414 cells. The grid must be 7×7 in size, and the ships must be placed either horizontally or vertically.
 3415 For example:

3416 . . AAA . .
 3417
 3418 C
 3419 C
 3420 . BBBB . C
 3421 C
 3422 C
 3423

3424 is a valid answer. The player must ensure that the ships do not overlap and that they fit within the
 3425 grid boundaries. The player will win if they correctly identify the locations of all three ships within
 3426 the allowed number of turns.

3427

3428 **(Hard 6) Quordle** Quordle is a word-guessing game where players attempt to deduce four hidden
 3429 8-letter words (all uppercase) simultaneously. Each guess provides feedback for each word: (1).Correct
 3430 letter in the correct position, represented by ‘A’. (2).Correct letter but misplaced, represented by
 3431 ‘M’. (3).Letter not in the word, represented by ‘X’. Players iteratively refine guesses using feedback
 3432 until solving all four words, within limited attempts. In each turn, players ask a single 8-letter up-
 3433 percase word, and will receive corresponding answer made up of ‘AMX’ for the four words. After
 3434 a few times of querying, the player will give four 8-letter uppercase words answers, separated by
 3435 commas. He will win if all four words are correct.

3436

3437 G.6 GAME STRATEGY INFERENCE (GSI)

3438 Easy black-boxes in GSI are typically with fixed game strategy or games with easy rules, while hard
 3439 black-boxes are with dynamic game strategy or games with difficult rules. Winning a round of game
 3440 will gain one point. A tie will gain zero point and a loss will cost one point. Such design of score
 3441 eliminates gain from randomness in a game. The expectation of a trivial strategy is supposed to
 3442 achieve zero point. In real-world scenarios with a finite number of game rounds, an LLM’s strategy
 3443 can sometimes result in a negative score. When this happens, we consider it a trivial strategy and
 3444 assign it a score of zero.

3445

3446 **(Easy 1) Load Shoot Defend Defender** In each turn, a player can choose ‘load’: gain one bullet,
 3447 ‘scout’: see the opponent’s bullet count, ‘shoot x’: spend ‘x’ bullets to attack, or ‘defend y’: spend
 3448 ‘y’ bullets to defend. Then, actions and the gaining points of both players are revealed, but the
 3449 specific numerical values ‘x’ and ‘y’ are kept hidden. Each player starts with 0 bullets and can hold
 3450 a maximum of 8. ‘x’ and ‘y’ cannot exceed the number of bullets they have currently. If one player
 3451 ‘shoot x’ and the other ‘defend y’, the defense succeeds and the player wins 1 point if $y \geq x$, and
 3452 the shooter wins -1 point. But if $y < x$, the attacker wins 1 point and the defender wins -1 point.
 3453 A Shoot action will always succeed against an opponent who chooses to Load or Scout. When a
 3454 simultaneous shoot happens, player with more bullets will win 1 point and the other win -1 point.
 3455 The same bullets will result in a tie where both players win 0 point. When a simultaneous defend
 or scout or load happens, both players win 0 point. If player ‘scout’, he will receive opponent’s
 bullet number. The game lasts for [total_turns] turns, both players try to maximize their scores. The

3456 black-box plays the role of an opponent with the following strategy: Cycle through ‘load’, ‘load’,
 3457 ‘defend 2’, ‘load’, ‘defend 1’,...
 3458

3459 **(Easy 2) RPS7 Cycle** Rock, paper, scissors, fire, water, air, sponge is a hand game where players
 3460 simultaneously choose between rock, paper, scissors, fire, water, air, sponge. Rock triumphs over
 3461 three opponents: it smashes Scissors, crushes Sponge, and puts out Fire. Paper also defeats three:
 3462 it covers Rock, floats on Water, and fans Air. Scissors have three victories: they cut Paper, shred
 3463 Sponge, and can create a spark to start a Fire. Fire is victorious against three as well: it melts
 3464 Scissors, burns Paper, and scorches Sponge. Water overcomes three challengers: it erodes Rock,
 3465 extinguishes Fire, and rusts Scissors. Air has three wins: it blows out Fire, erodes Rock, and evapo-
 3466 rates Water. Finally, Sponge defeats its three foes: it soaks up Water, cleans Paper (rendering it
 3467 useless), and utilizes its pockets to contain Air. In the game, two players repeat [total_turns] times,
 3468 each time the winner gets 1 point, the loser gets -1 point, and if they are tied, 0 point each. The
 3469 goal is to maximize total scores. The black-box plays the role of an opponent with the following
 3470 strategy: cycling through ‘rock’, ‘paper’, ‘scissors’, ‘fire’, ‘water’, ‘air’, ‘sponge’ in order, starting
 3471 with ‘rock’.

3472 **(Easy 3) RPS7 Random** Rock, paper, scissors, fire, water, air, sponge is a hand game where players
 3473 simultaneously choose between rock, paper, scissors, fire, water, air, sponge. Rock triumphs over
 3474 three opponents: it smashes Scissors, crushes Sponge, and puts out Fire. Paper also defeats
 3475 three: it covers Rock, floats on Water, and fans Air. Scissors have three victories: they cut Paper,
 3476 shred Sponge, and can create a spark to start a Fire. Fire is victorious against three as well: it
 3477 melts Scissors, burns Paper, and scorches Sponge. Water overcomes three challengers: it erodes
 3478 Rock, extinguishes Fire, and rusts Scissors. Air has three wins: it blows out Fire, erodes Rock, and
 3479 evaporates Water. Finally, Sponge defeats its three foes: it soaks up Water, cleans Paper (rendering
 3480 it useless), and utilizes its pockets to contain Air. In the game, two players repeat [total_turns] times,
 3481 each time the winner gets 1 point, the loser gets -1 point, and if they are tied, 0 point each. The goal
 3482 is to maximize total scores. The black-box plays the role of an opponent with the following strategy:
 3483 choose ‘rock’, ‘paper’, ‘air’ with the same probability.

3484 **(Easy 4) Anti RPS Random** Here we focus on a reversed version of Rock, Paper, Scissors. Rock,
 3485 Paper, Scissors is a hand game where players simultaneously choose between rock, paper, or scis-
 3486 sors, with rock crushing scissors, scissors cutting paper, and paper covering rock to determine the
 3487 winner. In the Anti Rock, Paper, Scissors game, the winning rules are exactly opposite to the origi-
 3488 nal ones. So scissors beat rock, paper beats scissors, and rock beats paper. In the game, two players
 3489 repeat [total_turns] times, each time the winner gets 1 point, the loser gets -1 point, and if they are
 3490 tied, 0 point each. The goal is to maximize total scores. The black-box plays the role of an opponent
 3491 with the following strategy: choose ‘rock’ and ‘scissors’ with the same probability.

3492 **(Easy 5) RPS7 Beat Last** Rock, paper, scissors, fire, water, air, sponge is a hand game where
 3493 players simultaneously choose between rock, paper, scissors, fire, water, air, sponge. Rock triumphs
 3494 over three opponents: it smashes Scissors, crushes Sponge, and puts out Fire. Paper also defeats
 3495 three: it covers Rock, floats on Water, and fans Air. Scissors have three victories: they cut Paper,
 3496 shred Sponge, and can create a spark to start a Fire. Fire is victorious against three as well: it
 3497 melts Scissors, burns Paper, and scorches Sponge. Water overcomes three challengers: it erodes
 3498 Rock, extinguishes Fire, and rusts Scissors. Air has three wins: it blows out Fire, erodes Rock, and
 3499 evaporates Water. Finally, Sponge defeats its three foes: it soaks up Water, cleans Paper (rendering it
 3500 useless), and utilizes its pockets to contain Air. In the game, two players repeat [total_turns] times,
 3501 each time the winner gets 1 point, the loser gets -1 point, and if they are tied, 0 point each. The goal
 3502 is to maximize total scores. The black-box plays the role of an opponent with the following
 3503 strategy: The first time you play a stone. And then each time you choose an action that can beat
 3504 your opponent’s last round action.

3505 **(Easy 6) RPS7 Imitate Last** Rock, paper, scissors, fire, water, air, sponge is a hand game where
 3506 players simultaneously choose between rock, paper, scissors, fire, water, air, sponge. Rock triumphs
 3507 over three opponents: it smashes Scissors, crushes Sponge, and puts out Fire. Paper also defeats
 3508 three: it covers Rock, floats on Water, and fans Air. Scissors have three victories: they cut Paper,
 3509 shred Sponge, and can create a spark to start a Fire. Fire is victorious against three as well: it

3510 melts Scissors, burns Paper, and scorches Sponge. Water overcomes three challengers: it erodes
 3511 Rock, extinguishes Fire, and rusts Scissors. Air has three wins: it blows out Fire, erodes Rock, and
 3512 evaporates Water. Finally, Sponge defeats its three foes: it soaks up Water, cleans Paper (rendering
 3513 it useless), and utilizes its pockets to contain Air. In the game, two players repeat [total_turns] times,
 3514 each time the winner gets 1 point, the loser gets -1 point, and if they are tied, 0 point each. The goal
 3515 is to maximize total scores. The black-box plays the role of an opponent with the following strategy:
 3516 first choose ‘fire’, second choose ‘air’, and then imitate your opponent’s behavior the time before
 3517 last.

3518

3519 **(Hard 1) Comparing Cards Smart** Comparing Cards is a game where two players play a card at
 3520 the same time and compare the number. Initially, both players have [total_cards] cards, numbered
 3521 from 1 to [total_cards]. In each turn, both players choose a card on his hand to play at the same time,
 3522 in the form of ‘card x’, where ‘x’ is the number on the card between 1 and [total_cards], and this
 3523 card must be available. The player with the higher number wins the turn and earns 1 point, while the
 3524 other player earns -1 point. If both players play the same number, then both players earn 0 points.
 3525 The game lasts for [total_cards] turns, both players try to maximize their scores. The black-box
 3526 plays the role of an opponent with the following strategy: First choose the card with median value.
 3527 If the opponent plays a card smaller than that, play accordingly to the order of ‘median-1, median-2,
 3528 ..., 1, median+1, median+2, ..., the highest value’. Else if the opponent plays a card larger than that,
 3529 play accordingly to the order of ‘median+1, median+2, ..., the highest value, median-1, median-2,
 3530 ..., 1’.

3530

3531 **(Hard 2) Comparing Cards Slice** Comparing Cards is a game where two players play a card at
 3532 the same time and compare the number. Initially, both players have [total_cards] cards, numbered
 3533 from 1 to [total_cards]. In each turn, both players choose a card on his hand to play at the same time,
 3534 in the form of ‘card x’, where ‘x’ is the number on the card between 1 and [total_cards], and this
 3535 card must be available. The player with the higher number wins the turn and earns 1 point, while the
 3536 other player earns -1 point. If both players play the same number, then both players earn 0 points.
 3537 The game lasts for [total_cards] turns, both players try to maximize their scores. The black-box
 3538 plays the role of an opponent with the following strategy: All the cards are arranged in ascending
 3539 order and divided equally into four piles, named a, b, c, and d, from smallest to largest. The cards
 3540 are played in the following sequence: Randomly select and play cards from pile b, one at a time,
 3541 until pile b is empty. Randomly select and play cards from pile d, one at a time, until pile d is empty.
 3542 Randomly select and play cards from pile a, one at a time, until pile a is empty. Randomly select
 3543 and play cards from pile c, one at a time, until pile c is empty.

3543

3544 **(Hard 3) Load Shoot Defend Smart** In each turn, a player can choose ‘load’: gain one bullet,
 3545 ‘scout’: see the opponent’s bullet count, ‘shoot x’: spend ‘x’ bullets to attack, or ‘defend y’: spend
 3546 ‘y’ bullets to defend. Then, actions and the gaining points of both players are revealed, but the
 3547 specific numerical values ‘x’ and ‘y’ are kept hidden. Each player starts with 0 bullets and can hold
 3548 a maximum of 8. ‘x’ and ‘y’ cannot exceed the number of bullets they have currently. If one player
 3549 ‘shoot x’ and the other ‘defend y’, the defense succeeds and the player wins 1 point if $y \geq x$, and
 3550 the shooter wins -1 point. But if $y < x$, the attacker wins 1 point and the defender wins -1 point.
 3551 A Shoot action will always succeed against an opponent who chooses to Load or Scout. When a
 3552 simultaneous shoot happens, player with more bullets will win 1 point and the other win -1 point.
 3553 The same bullets will result in a tie where both players win 0 point. When a simultaneous defend
 3554 or scout or load happens, both players win 0 point. If player ‘scout’, he will receive opponent’s
 3555 bullet number. The game lasts for [total_turns] turns, both players try to maximize their scores. The
 3556 black-box plays the role of an opponent with the following strategy: if the max turn is lower than
 3557 10, ‘load’ in the first 3 turns. Else, ‘load’ in the first 6 turns. If the opponent does not shoot in your
 3558 ‘load’ turns, keep ‘load’ until your opponent shoot first or you have 8 bullets. Then you keep ‘shoot
 3559 2’ until you don’t have enough bullets. Then ‘load’ until your opponent shoot first again. Then you
 3560 keep ‘shoot 2’ until you don’t have enough bullets, keep cycling this process.

3560

3561 **(Hard 4) RPS7 Mapping** Rock, paper, scissors, fire, water, air, sponge is a hand game where
 3562 players simultaneously choose between rock, paper, scissors, fire, water, air, sponge. Rock triumphs
 3563 over three opponents: it smashes Scissors, crushes Sponge, and puts out Fire. Paper also defeats
 three: it covers Rock, floats on Water, and fans Air. Scissors have three victories: they cut Paper,

shred Sponge, and can create a spark to start a Fire. Fire is victorious against three as well: it melts Scissors, burns Paper, and scorches Sponge. Water overcomes three challengers: it erodes Rock, extinguishes Fire, and rusts Scissors. Air has three wins: it blows out Fire, erodes Rock, and evaporates Water. Finally, Sponge defeats its three foes: it soaks up Water, cleans Paper (rendering it useless), and utilizes its pockets to contain Air. In the game, two players repeat [total_turns] times, each time the winner gets 1 point, the loser gets -1 point, and if they are tied, 0 point each. The goal is to maximize total scores. The black-box plays the role of an opponent with the following strategy: When last turn your opponent played ‘rock’, play ‘fire’. When last turn your opponent played ‘paper’, play ‘water’. When last turn your opponent played ‘scissors’, play ‘air’. When last turn your opponent played ‘fire’, play ‘sponge’. When last turn your opponent played ‘water’, play ‘rock’. When last turn your opponent played ‘air’, play ‘paper’. When last turn your opponent played ‘sponge’, play ‘scissors’.

(Hard 5) Load Shoot Defend Attacker In each turn, a player can choose ‘load’: gain one bullet, ‘scout’: see the opponent’s bullet count, ‘shoot x’: spend ‘x’ bullets to attack, or ‘defend y’: spend ‘y’ bullets to defend. Then, actions and the gaining points of both players are revealed, but the specific numerical values ‘x’ and ‘y’ are kept hidden. Each player starts with 0 bullets and can hold a maximum of 8. ‘x’ and ‘y’ cannot exceed the number of bullets they have currently. If one player ‘shoot x’ and the other ‘defend y’, the defense succeeds and the player wins 1 point if $y \geq x$, and the shooter wins -1 point. But if $y < x$, the attacker wins 1 point and the defender wins -1 point. A Shoot action will always succeed against an opponent who chooses to Load or Scout. When a simultaneous shoot happens, player with more bullets will win 1 point and the other win -1 point. The same bullets will result in a tie where both players win 0 point. When a simultaneous defend or scout or load happens, both players win 0 point. If player ‘scout’, he will receive opponent’s bullet number. The game lasts for [total_turns] turns, both players try to maximize their scores. The black-box plays the role of an opponent with the following strategy: Cycle through ‘load’, ‘load’, ‘shoot 2’ and ‘load’, ‘shoot 1’. i.e. repeat ‘load’, ‘load’, ‘shoot 2’, ‘load’, ‘shoot 1’, ‘load’, ‘load’, ‘shoot 2’, ...

(Hard 6) Load Shoot Defend Balance In each turn, a player can choose ‘load’: gain one bullet, ‘scout’: see the opponent’s bullet count, ‘shoot x’: spend ‘x’ bullets to attack, or ‘defend y’: spend ‘y’ bullets to defend. Then, actions and the gaining points of both players are revealed, but the specific numerical values ‘x’ and ‘y’ are kept hidden. Each player starts with 0 bullets and can hold a maximum of 8. ‘x’ and ‘y’ cannot exceed the number of bullets they have currently. If one player ‘shoot x’ and the other ‘defend y’, the defense succeeds and the player wins 1 point if $y \geq x$, and the shooter wins -1 point. But if $y < x$, the attacker wins 1 point and the defender wins -1 point. A Shoot action will always succeed against an opponent who chooses to Load or Scout. When a simultaneous shoot happens, player with more bullets will win 1 point and the other win -1 point. The same bullets will result in a tie where both players win 0 point. When a simultaneous defend or scout or load happens, both players win 0 point. If player ‘scout’, he will receive opponent’s bullet number. The game lasts for [total_turns] turns, both players try to maximize their scores. The black-box plays the role of an opponent with the following strategy: keep cycling ‘load’, ‘load’, ‘shoot 1’, ‘defend 1’, ...

H DETAILED EXPERIMENTAL RESULTS

We report detailed model performance of 19 benchmarked LLMs in 6 tasks in table form separately. The order of black-boxes remains the same as Section G.

3618
3619
3620

Table 10: Detailed model performance in the ORACLE benchmark 10@1&1@1.

Models	CII		CRI		PSI		ERI		IPI		GSI	
	Easy	Hard										
qwq-plus	37.45%	18.65%	30.00%	11.00%	48.72%	12.50%	3.41%	0.00%	43.33%	0.00%	28.18%	23.24%
qwen-plus*	44.20%	30.08%	48.00%	14.00%	50.00%	12.50%	3.41%	0.00%	70.00%	16.67%	31.06%	16.76%
qwen3-32b*	51.56%	16.03%	34.00%	12.00%	34.62%	6.25%	3.41%	0.00%	50.00%	5.56%	35.97%	26.02%
qwen3-235b*	49.05%	26.51%	32.00%	9.00%	47.44%	16.67%	5.68%	0.00%	46.67%	13.89%	58.76%	32.50%
o4-mini	77.23%	36.35%	48.00%	21.00%	61.54%	25.00%	63.64%	15.28%	80.00%	33.33%	62.68%	32.92%
o3-mini	56.71%	18.89%	36.00%	10.00%	32.05%	25.00%	38.64%	9.72%	46.67%	13.89%	54.73%	31.78%
o3	74.11%	52.14%	64.00%	24.00%	60.26%	43.75%	26.14%	15.28%	83.33%	36.11%	79.43%	43.95%
o1	36.02%	24.76%	44.00%	18.00%	34.62%	20.83%	21.59%	6.94%	40.00%	22.22%	51.14%	25.43%
gemini-2.5-pro	72.77%	39.29%	54.00%	23.00%	53.85%	41.67%	17.05%	1.39%	73.33%	33.33%	61.60%	46.76%
gemini-2.5-flash*	54.98%	21.83%	20.00%	13.00%	41.03%	31.25%	3.41%	0.00%	30.00%	19.44%	42.19%	36.46%
gemini-2.5-flash	57.27%	27.22%	20.00%	10.00%	28.21%	18.75%	3.41%	0.00%	13.33%	2.78%	14.08%	22.18%
gemini-2.0-flash	38.18%	14.52%	18.00%	18.00%	6.41%	6.25%	2.27%	0.00%	10.00%	2.78%	16.42%	19.17%
deepseek-r1	45.71%	39.05%	44.00%	17.00%	53.85%	16.67%	1.14%	0.00%	53.33%	19.44%	52.60%	25.33%
deepseek-v3	58.74%	19.76%	24.00%	12.00%	19.23%	2.08%	1.14%	0.00%	16.67%	0.00%	17.58%	10.13%
claude-4-sonnet*	64.03%	26.35%	44.00%	19.00%	43.59%	12.50%	5.68%	0.00%	53.33%	30.56%	47.59%	27.84%
claude-4-sonnet	53.33%	23.17%	32.00%	13.00%	15.38%	2.08%	4.55%	0.00%	43.33%	13.89%	32.27%	17.33%
claude-3.7-sonnet*	68.66%	27.22%	52.00%	12.00%	48.72%	35.42%	22.73%	5.56%	40.00%	27.78%	39.96%	25.47%
claude-3.7-sonnet	46.93%	22.54%	18.00%	12.00%	33.33%	10.42%	9.09%	0.00%	30.00%	5.56%	41.52%	14.75%
claude-3.5-sonnet	51.73%	16.51%	26.00%	17.00%	24.36%	12.50%	6.82%	0.00%	26.67%	5.56%	38.51%	20.98%

3634
3635
3636
3637
3638

Table 11: Detailed model performance in the ORACLE benchmark 20@2&2@1.

Models	CII		CRI		PSI		ERI		IPI		GSI	
	Easy	Hard										
qwq-plus	62.60%	38.89%	34.00%	12.00%	48.72%	14.58%	3.41%	0.00%	60.00%	2.78%	25.93%	18.30%
qwen-plus*	61.13%	42.30%	70.00%	30.00%	43.59%	18.75%	3.41%	0.00%	76.67%	22.22%	38.99%	16.29%
qwen3-32b*	67.88%	37.62%	50.00%	10.00%	29.49%	16.67%	6.82%	0.00%	83.33%	13.89%	38.46%	38.36%
qwen3-235b*	77.10%	34.76%	56.00%	16.00%	47.44%	22.92%	4.55%	0.00%	76.67%	16.67%	62.82%	51.40%
o4-mini	92.60%	58.02%	76.00%	35.00%	64.10%	35.42%	61.36%	16.67%	86.67%	44.44%	55.11%	23.21%
o3-mini	75.11%	34.76%	52.00%	19.00%	41.03%	29.17%	43.18%	11.11%	76.67%	16.67%	49.50%	34.22%
o3	90.52%	74.84%	84.00%	47.00%	65.38%	37.50%	51.14%	5.56%	96.67%	66.67%	82.51%	54.79%
o1	45.28%	24.84%	62.00%	17.00%	42.31%	20.83%	12.50%	0.00%	83.33%	30.56%	49.92%	34.11%
gemini-2.5-pro	88.05%	57.30%	84.00%	52.00%	65.38%	31.25%	26.14%	16.67%	96.67%	44.44%	60.77%	41.02%
gemini-2.5-flash*	71.90%	35.08%	44.00%	31.00%	35.90%	16.67%	6.82%	0.00%	80.00%	36.11%	52.83%	26.89%
gemini-2.5-flash	60.48%	23.81%	34.00%	11.00%	35.90%	22.92%	6.82%	0.00%	20.00%	5.56%	31.48%	19.51%
gemini-2.0-flash	39.61%	5.87%	44.00%	21.00%	0.00%	2.08%	4.55%	1.39%	13.33%	2.78%	17.96%	16.39%
deepseek-r1	73.77%	43.73%	70.00%	27.00%	50.00%	31.25%	3.41%	0.00%	63.33%	41.67%	45.71%	24.58%
deepseek-v3	70.48%	23.17%	44.00%	21.00%	20.51%	6.25%	3.41%	0.00%	40.00%	0.00%	20.54%	14.34%
claude-4-sonnet*	84.03%	48.33%	56.00%	41.00%	55.13%	27.08%	23.86%	1.39%	83.33%	41.67%	62.97%	36.27%
claude-4-sonnet	71.99%	34.44%	42.00%	22.00%	28.21%	4.17%	4.55%	0.00%	60.00%	16.67%	44.31%	20.28%
claude-3.7-sonnet*	78.66%	46.59%	64.00%	34.00%	56.41%	37.50%	18.18%	0.00%	80.00%	38.89%	49.54%	32.36%
claude-3.7-sonnet	69.18%	37.30%	52.00%	26.00%	34.62%	12.50%	11.36%	0.00%	56.67%	16.67%	41.61%	17.78%
claude-3.5-sonnet	64.42%	33.65%	36.00%	20.00%	34.62%	10.42%	7.95%	0.00%	60.00%	8.33%	32.69%	8.73%

3652
3653
3654
3655
3656

Table 12: Detailed model performance on CII task in the ORACLE benchmark 10@1.

Models	Easy Black-Box in CII											Hard Black-Box in CII					
	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6
qwq-plus	0.23	0.54	0	0.3	0.33	0.11	0.2	0.9	0.47	0.43	0.6	0	0.14	0.2	0.26	0.09	0.43
qwen-plus*	0.53	0.43	0.37	0.43	0.27	0.11	0.37	0.97	0.4	0.89	0.63	0.4	0.4	0	0.17	0.29	0.33
qwen3-32b*	0.37	0.63	0	0.43	0.3	0.09	0.3	0.87	0.57	0.66	0.6	0.2	0.31	0.3	0.37	0.29	0.33
qwen3-235b*	0.5	0.46	0.54	0.53	0.5	0.43	0.07	0.97	0.3	0.63	0.74	0.06	0.23	0.1	0.11	0.23	0.23
o4-mini	0.97	0.91	0.54	0.87	0.8	0.66	0.83	0.97	0.63	0.6	0.71	0.37	0.4	0.23	0.23	0.31	0.63
o3-mini	0.57	0.77	0.69	0.33	0.23	0.37	0.9	0.53	0.5	0.89	0.46	0.2	0.09	0.17	0.03	0.29	0.37
o3	0.93	0.8	0.29	0.77	0.87	0.54	1	0.93	0.57	0.74	0.71	0.54	0.49	0.5	0.51	0.29	0.8
o1	0.8	0.11	0.26	0	0.03	0.49	0.3	0.5	0.1	0.77	0.6	0.4	0.34	0.13	0.2	0.14	0.27
gemini-2.5-pro	0.73	0.77	0.63	0.57	0.77	0.4	0.97	0.87	0.73	0.86	0.71	0.63	0.31	0.23	0.26	0.67	0.67
gemini-2.5-flash*	0.27	0.66	0.23	0.47	0.47	0.49	1	0.6	0.53	0.66	0.69	0	0.31	0.2	0.23	0.2	0.37
gemini-2.5-flash	0.57	0.8	0.29	0.57	0.6	0	1	0.63	0.73	0.46	0.66	0.23	0.37	0.2	0.26	0.14	0.43
gemini-2.0-flash	0.13	0.34	0.46	0.53	0.3	0.57	0	0.53	0.3	0.57	0.46	0.09	0.54	0.07	0	0.14	0.03
deepseek-r1	0.07	0.11	0.43	0.33	0.53	0.49	0.43	0.73	0.5	0.86	0.54	0.46	0.29	0.27	0.23	0.73	0.73
deepseek-v3	0.7	0.57	0.34	0.6	0.6	0.66	0.8	0.67	0.47	0.6	0.46	0.11	0.29	0.03	0.34	0.14	0.27
claude-4-sonnet*	0.67	0.91	0.69	0.53	0	0	1	1	0.5	0.94	0.8	0.34	0.37	0.23	0.37	0.03	0.23
claude-4-sonnet	0.43	0.6	0.26	0.5	0.43	0.51	1	0.53	0.57	0.57	0.46	0.23	0.46	0.27	0.17	0	0.27
claude-3.7-sonnet*	0.37	0.77	0.57	0.57	0.67	0.51	0.93	0.97	0.77	0.89	0.54	0.09	0.34	0.1	0.26	0.51	0.33
claude-3.7-sonnet	0.13	0.57	0.57	0.53	0.5	0.57	0.33	0.53	0.5	0.46	0.46	0.31	0.23	0.1	0.2	0.14	0.37
claude-3.5-sonnet	0.3	0.34	0.34	0.5	0.33	0.54	0.97	0.5	0.63	0.57	0.66	0.23	0.23	0.13	0.26	0.14	0

3671

3672
 3673
 3674
 3675
 3676

Table 13: Detailed model performance on CII task in the ORACLE benchmark 20@2.

Models	Easy Black-Box in CII											Hard Black-Box in CII					
	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6
qwq-plus	0.34	0.94	0.83	0.53	0.37	0.23	0.43	0.97	0.5	0.97	0.77	0.37	0.6	0.2	0.34	0.29	0.53
qwen-plus*	0.2	0.91	0.43	0.53	0.8	0.34	0.53	1	0.4	0.8	0.77	0.51	0.63	0.3	0.37	0.29	0.43
qwen3-32b*	0.43	0.94	0.86	0.57	0.67	0.51	0.27	0.97	0.6	0.86	0.8	0.34	0.53	0.27	0.29	0.43	0.4
qwen3-235b*	0.6	0.46	0.89	0.63	0.87	0.6	0.9	1	0.77	1	0.77	0.34	0.5	0.27	0.23	0.31	0.43
o4-mini	0.77	0.94	1	0.9	1	0.91	1	1	0.8	0.91	0.94	0.31	0.93	0.3	0.54	0.46	0.93
o3-mini	0.37	0.86	0.83	0.63	0.7	0.66	1	0.9	0.6	0.91	0.8	0.6	0.43	0.37	0.2	0.29	0.2
o3	0.66	1	1	0.9	1	0.77	1	1	0.8	0.83	1	0.74	1	0.67	0.77	0.34	0.97
o1	0.63	0.69	1	0.2	0	0	0.13	0.87	0.07	0.83	0.57	0.43	0.57	0.03	0.11	0.31	0.03
gemini-2.5-pro	0.94	0.94	0.94	0.63	0.97	0.57	1	0.93	0.87	0.94	0.94	0.69	0.77	0.3	0.46	0.43	0.8
gemini-2.5-flash*	0.46	0.97	0.43	0.63	0.83	0.43	1	0.8	0.9	0.74	0.71	0	0.5	0.3	0.34	0.43	0.53
gemini-2.5-flash	0.46	0	0.49	0.6	0.9	0.43	1	0.73	0.73	0.63	0.69	0	0.63	0.17	0.46	0.17	0
gemini-2.0-flash	0.14	0.71	0	0.53	0.4	0.43	0.13	0.63	0.4	0.51	0.46	0.09	0.03	0.03	0	0.2	0
deepseek-r1	0.74	0.91	0.54	0.83	0.97	0.37	0.8	0.73	0.67	0.8	0.74	0.49	0.77	0.37	0.6	0.17	0.23
deepseek-v3	0.6	0.77	0.54	0.7	0.8	0.66	1	0.67	0.7	0.71	0.6	0.26	0.5	0.23	0.26	0.14	0
claude-4-sonnet*	0.74	0.94	0.91	0.67	0.9	0.63	1	0.9	0.83	0.91	0.8	0.51	0.77	0.33	0.34	0.14	0.8
claude-4-sonnet	0.63	0.91	0.86	0.63	0.6	0.51	1	0.67	0.73	0.71	0.66	0.37	0.53	0.2	0.29	0.14	0.53
claude-3.7-sonnet*	0.43	0.94	0.89	0.6	0.9	0.57	1	0.97	0.9	0.66	0.8	0.49	0.5	0.2	0.37	0.77	0.47
claude-3.7-sonnet	0.54	0.69	0.71	0.63	0.8	0.66	1	0.7	0.73	0.46	0.69	0.51	0.63	0.17	0.29	0.17	0.47
claude-3.5-sonnet	0.6	0.66	0.6	0.7	0.6	0.71	1	0.63	0.47	0.46	0.66	0.43	0.57	0.13	0.31	0.14	0.43

3692
 3693
 3694
 3695
 3696
 3697
 3698
 3699
 3700
 3701

Table 14: Detailed model performance on CRI task in the ORACLE benchmark 10@1.

Models	Easy Black-Box in CRI					Hard Black-Box in CRI									
	1	2	3	4	5	1	2	3	4	5	6	7	8	9	10
qwq-plus	0.5	0.7	0.3	0	0	0.2	0	0.3	0	0	0	0.4	0	0.2	0
qwen-plus*	1	0.6	0.7	0.1	0	0.1	0.1	0.3	0.1	0.1	0	0.7	0	0	0
qwen3-32b*	0.8	0.6	0.1	0	0.2	0	0.1	0.5	0.1	0.1	0	0.4	0	0	0
qwen3-235b*	0.8	0.7	0.1	0	0	0	0	0.2	0.1	0	0	0.6	0	0	0
o4-mini	0.6	0.8	0.8	0	0.2	0	0.1	0.4	0	0.1	0	1	0	0.5	0
o3-mini	0.9	0.6	0.1	0	0.2	0.4	0	0.3	0	0	0	0.2	0	0.1	0
o3	0.9	0.9	0.6	0.6	0.2	0.2	0.1	0.5	0	0.1	0	0.7	0.2	0.6	0
o1	0.8	0.8	0.4	0	0.2	0.2	0	0.5	0.1	0.2	0	0.8	0	0	0
gemini-2.5-pro	0	0.9	1	0.5	0.3	0.1	0	0.4	0.1	0.2	0	1	0.2	0.3	0
gemini-2.5-flash*	0	0.8	0.1	0.1	0	0.2	0	0.5	0	0.1	0	0.4	0.1	0	0
gemini-2.5-flash	0	0.8	0.2	0	0	0.1	0	0.3	0	0.2	0	0.4	0	0	0
gemini-2.0-flash	0	0	0.2	0.1	0.6	0.1	0.3	0.5	0	0.4	0.2	0.2	0	0.1	0
deepseek-r1	0.9	0.7	0.3	0.1	0.2	0	0.1	0.4	0.2	0.2	0	0.4	0	0.4	0
deepseek-v3	0	0.1	0.5	0.2	0.4	0.2	0.1	0.6	0.2	0	0	0.1	0	0	0
claude-4-sonnet*	0.9	0.9	0.1	0.2	0.1	0.2	0.1	0.4	0.1	0.2	0	0.5	0.3	0.1	0
claude-4-sonnet	0.9	0.5	0	0	0.2	0.1	0.2	0.4	0.1	0.2	0	0.3	0	0	0
claude-3.7-sonnet*	0.7	0.9	0.4	0	0.6	0.1	0	0.5	0	0.2	0	0.3	0.1	0	0
claude-3.7-sonnet	0	0.5	0.2	0	0.2	0.1	0.1	0.5	0.2	0.3	0	0	0	0	0
claude-3.5-sonnet	0	0.7	0.3	0.1	0.2	0.1	0.1	0.5	0.2	0.3	0	0.5	0	0	0

3722
 3723
 3724
 3725

3726

3727

Table 15: Detailed model performance on CRI task in the ORACLE benchmark 20@2.

3728

3729

Models	Easy Black-Box in CRI					Hard Black-Box in CRI									
	1	2	3	4	5	1	2	3	4	5	6	7	8	9	10
qwq-plus	0.9	0.7	0.1	0	0	0	0	0.4	0	0	0	0.8	0	0	0
qwen-plus*	1	0.9	1	0.1	0.5	0.7	0.3	0.7	0	0.2	0	1	0	0.1	0
qwen3-32b*	0.9	0.6	0.7	0.1	0.2	0.1	0	0.4	0	0	0	0.5	0	0	0
qwen3-235b*	1	0.9	0.6	0	0.3	0	0	0.5	0.1	0.1	0	0.8	0	0.1	0
o4-mini	1	1	1	0.3	0.5	0.8	0.1	0.5	0	0.3	0	1	0.1	0.7	0
o3-mini	1	1	0.5	0	0.1	0.6	0	0.4	0	0	0	0.9	0	0	0
o3	1	1	1	0.8	0.4	0.8	0.3	0.8	0.2	0.3	0	1	0.3	0.9	0.1
o1	1	1	0.6	0	0.5	0.4	0	0.6	0.1	0.1	0	0.4	0	0.1	0
gemini-2.5-pro	1	1	1	1	0.2	0.9	0.5	0.5	0.2	0.3	0	1	0.7	1	0.1
gemini-2.5-flash*	0.2	0.9	0.9	0	0.2	0.6	0.3	0.2	0.2	0.2	0	0.6	0.7	0.3	0
gemini-2.5-flash	0	1	0.7	0	0	0	0.3	0.3	0	0.2	0.1	0.2	0	0	0
gemini-2.0-flash	0.1	1	0.3	0.2	0.6	0.3	0.5	0.5	0	0.3	0.3	0.1	0	0	0.1
deepseek-r1	1	1	1	0.1	0.4	0.1	0.1	0.5	0.2	0.3	0	1	0	0.5	0
deepseek-v3	1	0.4	0.2	0.2	0.4	0.4	0.1	0.6	0.2	0.1	0	0.4	0	0.3	0
claude-4-sonnet*	1	1	0.4	0.1	0.3	0.6	0.2	0.5	0.2	0.2	0.1	0.9	0.7	0.7	0
claude-4-sonnet	0.1	1	0.4	0.4	0.2	0.4	0.1	0.5	0.1	0.4	0.1	0.6	0	0	0
claude-3.7-sonnet*	1	1	0.7	0.1	0.4	0.7	0.2	0.5	0	0.5	0.3	1	0.2	0	0
claude-3.7-sonnet	0.6	1	0.4	0.4	0.2	0.4	0.2	0.5	0.2	0.7	0.1	0.5	0	0	0
claude-3.5-sonnet	0	1	0.4	0.2	0.2	0.3	0.3	0.5	0	0.4	0.2	0.3	0	0	0

3746

3747

3748

3749

Table 16: Detailed model performance on PSI task in the ORACLE benchmark 10@1.

3750

Models	Easy Black-Box in PSI													Hard Black-Box in PSI							
	1	2	3	4	5	6	7	8	9	10	11	12	13	1	2	3	4	5	6	7	8
qwq-plus	1	1	0.33	0	1	0	0.17	0	0	0.17	1	1	0.67	0	0	0.33	0	0	0	0.67	0
qwen-plus*	1	0.83	0	0	1	0	0.33	0	0.17	0.17	1	1	0.17	0.17	0	0.33	0	0	0	0.33	0
qwen3-32b*	1	0	0.17	0	1	0	0	0	0	0	1	1	0.33	0	0.17	0	0.33	0	0	0	0
qwen3-235b*	1	0.67	0.33	0	1	0	0	0.17	0	0	1	1	0	0.5	0	0.5	0	0	0	0.33	0
o4-mini	1	0.33	0.33	1	1	0.17	0.33	0	0.83	1	0.83	1	0.17	0	0.5	0.67	0.17	0	0	0.67	0
o3-mini	0.17	0	0.17	0	1	0.17	0.33	0	0	0.17	1	1	0.17	0	0.17	0.5	0.5	0	0.17	0	0.67
o3	1	0.83	0.33	0	1	0.17	0.5	0.67	0.33	0.17	1	1	0.83	0.5	0.67	0.5	0.67	0.17	0.17	0.83	0
o1	0.17	0.5	0.17	0.17	1	0.17	0.33	0	0	0	0.83	1	0.17	0	0.5	0.5	0.33	0	0.17	0	0.17
gemini-2.5-pro	1	1	0.33	0	1	0.17	0.17	0.17	0.17	0.17	1	1	0.83	0.33	0.83	0.5	0.67	0	0.33	0	0.67
gemini-2.5-flash*	0.33	0.67	0.17	0	0.83	0.17	0.17	0	0	0.17	1	1	0.83	0.17	0.67	0.33	0.17	0	0.17	0	0.5
gemini-2.5-flash	0	0.17	0	0.17	0.33	0	0.17	0.17	0	0	0.17	1	1	0.5	0.17	0.67	0.33	0.17	0	0	0.17
gemini-2.0-flash	0.33	0	0.17	0	0.17	0	0.17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
deepseek-r1	0.83	1	0	0	1	0	0	0.33	0.67	0.17	1	1	1	0	0	0	0.5	0.17	0	0	0.33
deepseek-v3	0.17	0	0	0.33	0	0	0	0	0	0	1	1	0	0	0.17	0	0	0	0	0	0
claude-4-sonnet*	0.83	0.33	0	0	0	1	0.33	0.17	0.17	0.17	1	1	0.67	0	0.17	0.33	0.33	0	0	0	0.17
claude-4-sonnet	0.17	0	0	0	0.17	0	0	0	0	0	0.83	0.83	0	0	0	0	0	0.17	0	0	0
claude-3.7-sonnet*	0.83	0.33	0.17	0	1	0.17	0.33	0.33	0.17	0.17	1	1	0.83	0.17	0.67	0.5	0.67	0	0.17	0	0.67
claude-3.7-sonnet	0.33	0.17	0	0.17	0.33	0.17	0.17	0.17	0.17	0.17	1	1	0.5	0	0.17	0	0.33	0	0	0	0.33
claude-3.5-sonnet	0.33	0.17	0	0.17	0.33	0	0	0	0	0	0.83	1	0.33	0	0.33	0	0.33	0	0	0	0.33

3763

3764

3765

3766

Table 17: Detailed model performance on PSI task in the ORACLE benchmark 20@2.

3767

Models	Easy Black-Box in PSI													Hard Black-Box in PSI							
	1	2	3	4	5	6	7	8	9	10	11	12	13	1	2	3	4	5	6	7	8
qwq-plus	0.5	1	1	0	0.83	0	0.33	0	0	0.17	0.83	1	0.67	0	0	0	0.67	0	0	0.17	0.33
qwen-plus*	0.17	0	0	0.17	1	0	0.67	0.67	0	0.5	1	1	0.5	0.17	0.5	0.33	0.33	0	0	0	0.17
qwen3-32b*	0.67	0.5	0	0.17	1	0	0.17	0	0	0	0	1	0.33	0	0	0.33	0.33	0	0	0	0.67
qwen3-235b*	0.83	0.17	1	0.17	1	0	0.17	0	0	0.5	1	1	0.33	0.17	0.17	0.5	0.67	0	0	0	0.33
o4-mini	0.83	1	0.17	0	1	0	0.5	0.83	0.17	0.83	1	1	1	0	0.83	0.83	0.67	0	0	0	0.5
o3-mini	0.67	0	0.17	0.17	1	0	0	0.5	0.33	0.33	1	1	0.17	0	1	0.17	0.67	0	0	0	0.5
o3	1	1	0.17	0.17	1	0	0.5	1	0.5	0.33	1	1	0.83	0	0.83	0.67	0.67	0	0	0	0.83
o1	0.33	0.83	0.17	0.17	1	0	0.17	0.5	0.17	0	1	1	0	0	0.17	0.5	0.33	0	0	0	0.67
gemini-2.5-pro	1	1	0.17	0.17	1	0	0.33	0.5	0.33	1	1	1	1	0	0	0	0.5	0.83	0	0.33	0.83
gemini-2.5-flash*	0.83	0.17	0.17	0.17	0.5	0.17	0.17	0	0.17	1	1	0.17	0	0	0	0.33	0.83	0	0	0	0.17
gemini-2.5-flash	0.5	0.33	0.17	0.5	0.17	0.17	0	0	0.17	1	1	0.5	0.17	0.5	0.33	0.67	0	0	0	0.17	0.17
gemini-2.0-flash	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
deepseek-r1	1	1	0.17	0	1	0	0.17	0	0.17	0.83	1	1	0.5	0.67	0	0.67	0	0	0	0	0
deepseek-v3	0.17	0	0	0.17	0	0	0.17	0.17	0	0.17	0.83	1	0	0	0.17	0.17	0.17	0	0	0	0
claude-4-sonnet*	1	0.5	0.33	0.17	1	0.17	0.33	0.33	0.17	0.33	1	1	0.83	0	0.5	0.33	0.67	0	0	0	0.67
claude-4-sonnet	0.33	0	0.17	0.17	0.17	0.17	0.17	0	0.17	1	1	0.33	0	0	0	0.17	0	0	0	0	0.17
claude-3.7-sonnet*	1	0.67	0.17	0.17	1	0.17	0.33	0.5	0.17	0.17	1	1	1	0.33	0.67	0.5	0.83	0	0	0	0.67
claude-3.7-sonnet	0.33	0.17	0.17	0.17	0.5	0.17	0.17	0.17	0.17	0.17	0.83	1	0.5	0.33	0	0.17	0.33	0			

Table 18: Detailed model performance on ERI task in the ORACLE benchmark 10@1.

Models	Easy Black-Box in ERI											Hard Black-Box in ERI								
	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	9
qwq-plus	0.13	0	0.13	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0
qwen-plus*	0.13	0	0.13	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0
qwen3-32b*	0	0.13	0.38	0	0	0	0	0	0	0.13	0.13	0	0	0	0	0	0	0	0	0
qwen3-235b*	0.25	0	0	0	0	0	0.13	0.13	0	0	0	0	0	0	0	0	0	0	0	0
o4-mini	0.25	0.25	0.75	0	0	1	0.75	1	0.75	1	1	0	0	0	0	0	0	0	1	0.5
o3-mini	0.25	0	0.25	0.38	0	0.88	0.13	0.13	0.75	1	1	0	0	0	0	0	0	0	1	0
o3	0.13	0	1	0.38	0	0	0.38	1	1	0.75	1	0	0	0	0	0	0	0	0	0.5
o1	0	0.38	0	0	0	0.75	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0
gemini-2.5-pro	0.875	1	0.25	0.13	0	0	0.13	0.5	0.13	1	0.38	0	0	0.13	0.13	0.5	0	0	0	0
gemini-2.5-flash*	0.25	0	0	0	0	0	0	0	0.38	0.13	0	0	0	0	0	0	0	0	0	0
gemini-2.5-flash	0.13	0	0	0	0	0	0.13	0.13	0	0.38	0	0	0	0	0	0	0	0	0	0
gemini-2.0-flash	0.25	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0.13	0	0	0	0	0
deepseek-r1	0	0	0	0	0.25	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
deepseek-v3	0.25	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
claude-4-sonnet*	0.25	0.13	0.38	0.75	0	0	0.13	0	0.13	0.13	0.75	0	0	0	0.13	0	0	0	0	0
claude-4-sonnet	0.25	0	0	0	0	0	0.13	0.13	0	0	0	0	0	0	0	0	0	0	0	0
claude-3.7-sonnet*	0.25	0	0.38	0.63	0	0	0.13	0	0.38	0.13	0.13	0	0	0	0	0	0	0	0	0
claude-3.7-sonnet	0.25	0	0	0	0	0.13	0	0.38	0.13	0	0	0	0	0	0	0	0	0	0	0
claude-3.5-sonnet	0.25	0	0	0	0	0.13	0	0.38	0.13	0	0	0	0	0	0	0	0	0	0	0

Table 19: Detailed model performance on ERI task in the ORACLE benchmark 20@2.

Models	Easy Black-Box in ERI											Hard Black-Box in ERI								
	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	9
qwq-plus*	0.13	0	0.13	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0
qwen-plus*	0.13	0	0.13	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0
qwen3-32b*	0.13	0	0.13	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0
qwen3-235b*	0.25	0	0.13	0	0	0	0.13	0.13	0	0	0	0	0	0	0	0	0	0	0	0
o4-mini	0.75	0.5	0.25	1	0	1	0.75	0.63	0.88	0.25	1	0.13	0	0.13	0.13	1	0	0	0	0
o3-mini	0	0.38	0.25	0.88	0	0.25	0.75	0.75	0	0	1	0	0	0	0.13	0	0.75	0	0	0
o3	0.5	0.13	0.25	0	0	0.38	0.75	0.88	0	0	0	0	0	0	0	0.38	1	0	0	0
o1	0	0	0	0.5	0	0	0.13	0	0	1	0.75	0	0	0	0	0	0	0.63	0	0
gemini-2.5-pro	0.88	0	0.13	0.13	0	0.38	0.25	0.13	0	0	0	0	0	0.13	0	0	0	0	0	0
gemini-2.5-flash*	0.13	0	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
gemini-2.5-flash	0	0	0.25	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0
gemini-2.0-flash	0	0	0	0	0	0	0.13	0	0	0.13	0	0	0	0	0	0	0	0	0	0
deepseek-r1	0	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0
deepseek-v3	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0
claude-4-sonnet*	0	0	0.13	0.13	0	0.25	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0
claude-4-sonnet	0.13	0	0.25	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0
claude-3.7-sonnet*	0.25	0	0.13	0.13	0	0	0.13	0.13	0.75	0.75	0.75	0.13	0.13	0	0	0	0	0.25	0	0
claude-3.7-sonnet	0	0	0.25	0	0	0	0	0	0.13	0	0.63	0	0	0	0	0	0	0	0	0
claude-3.5-sonnet	0	0	0.25	0	0	0	0	0	0.13	0	0.38	0	0	0	0	0	0	0	0	0

Table 20: Detailed model performance on IPI task in the ORACLE benchmark 10@1.

Models	Easy Black-Box in IPI					Hard Black-Box in IPI					
	1	2	3	4	5	1	2	3	4	5	6
qwq-plus	0.17	0.83	0.67	0	0.5	0	0	0	0	0	0
qwen-plus*	0.67	1	0.67	0.5	0.67	0	0.83	0	0.17	0	0
qwen3-32b*	0.17	1	0.83	0	0.5	0	0	0	0.33	0	0
qwen3-235b*	0.17	0.83	0.5	0.5	0.33	0.17	0.5	0	0.17	0	0
o4-mini	0.17	1	0.83	1	1	0	1	0.17	0.17	0.83	0
o3-mini	0	0.67	0.67	0.33	0.67	0	0.17	0	0.67	0	0
o3	0.17	1	1	1	1	0.33	1	0.17	0.67	0	0
o1	0.17	0.67	0.67	0.17	0.33	0	0.5	0	0.83	0	0
gemini-2.5-pro	0.33	0.83	0.83	0.83	0.83	0.5	1	0.17	0.33	0	0
gemini-2.5-flash*	0.17	0.5	0.5	0	0.33	0	0.67	0	0.5	0	0
gemini-2.5-flash	0.17	0.33	0.17	0	0	0	0	0	0.17	0	0
gemini-2.0-flash	0.17	0.33	0	0	0	0	0	0	0.17	0	0
deepseek-r1	0.33	0.67	1	0.33	0.33	0	0.67	0.17	0.33	0	0
deepseek-v3	0.17	0.67	0	0	0	0	0	0	0	0	0
claude-4-sonnet*	0.33	0.83	0.5	0.83	0.17	0	1	0	0.83	0	0
claude-4-sonnet	0.67	0.83	0.67	0	0	0	0	0	0.83	0	0
claude-3.7-sonnet*	0.33	0.67	0.17	0.83	0	0	0.67	0	1	0	0
claude-3.7-sonnet	0.5	0.83	0.17	0	0	0	0	0	0.33	0	0
claude-3.5-sonnet	0.17	0.83	0.33	0	0	0	0	0	0.33	0	0

3834

3835

3836

3837

Table 21: Detailed model performance on IPI task in the ORACLE benchmark 20@2.

Models	Easy Black-Box in IPI					Hard Black-Box in IPI					
	1	2	3	4	5	1	2	3	4	5	6
qwq-plus	1	1	0	0.67	0.33	0	0	0	0.17	0	0
qwen-plus*	1	1	0.5	0.83	0.5	0.5	0.67	0	0.17	0	0
qwen3-32b*	0.83	1	0.83	1	0.5	0	0.17	0	0.67	0	0
qwen3-235b*	1	1	0.5	1	0.33	0.5	0.33	0	0.17	0	0
o4-mini	1	0.83	0.83	1	0.67	0.5	1	0.33	0.83	0	0
o3-mini	1	1	0.67	1	0.17	0	0.5	0	0.5	0	0
o3	1	1	1	1	0.83	0.83	1	1	1	0.17	0
o1	1	1	0.5	0.83	0.83	0.5	0.5	0	0.83	0	0
gemini-2.5-pro	1	1	1	1	0.83	1	1	0.17	0.5	0	0
gemini-2.5-flash*	0.83	1	0.67	1	0.5	0.5	0.5	0.17	1	0	0
gemini-2.5-flash	0.67	0.17	0	0	0.17	0	0	0	0.33	0	0
gemini-2.0-flash	0.33	0.17	0	0	0.17	0	0	0	0.17	0	0
deepseek-r1	1	1	0.33	0.5	0.33	0.83	0.67	0	0.83	0.17	0
deepseek-v3	0.83	0.67	0	0	0.5	0	0	0	0	0	0
claude-4-sonnet*	1	1	0.83	0.33	1	0.5	1	0	1	0	0
claude-4-sonnet	1	1	0	0	1	0	0	0	1	0	0
claude-3.7-sonnet*	1	1	0.67	0.5	0.83	0.17	0.83	0	1	0.33	0
claude-3.7-sonnet	0.83	0.83	0	0.33	0.83	0	0.17	0	0.83	0	0
claude-3.5-sonnet	1	1	0.17	0.17	0.67	0	0	0	0.5	0	0

3859

3860

3861

3862

3863

3864

3865

Table 22: Detailed model performance on GSI task in the ORACLE benchmark 1@1.

Models	Easy Black-Box in GSI						Hard Black-Box in GSI					
	1	2	3	4	5	6	1	2	3	4	5	6
qwq-plus	0.27	0.65	0	0.39	0.11	0.27	0.33	0.68	0	0.39	0	0
qwen-plus*	0.75	0.85	0.82	0.20	0.74	0.16	0.34	0.58	0.25	0.42	0	0.36
qwen3-32b*	0.32	0.44	0.31	0.45	0.18	0.16	0.15	0.14	0	0.52	0	0.19
qwen3-235b*	0.08	0.69	0.50	0.48	0.28	0.14	0.47	0.29	0	0.47	0	0.34
o4-mini	0.76	0.84	1	0.58	0.26	0.32	0.41	0.45	0.42	0.38	0.1	0.22
o3-mini	0.82	0.82	0.83	0.36	0	0.46	0.33	0.54	0.23	0.7	0.11	0
o3	0.94	0.91	0.93	0.71	0.79	0.49	0.49	0.33	0.70	0.4	0.21	0.51
o1	0.41	0.70	0.71	0.97	0.09	0.19	0.44	0.33	0	0.46	0.20	0.11
gemini-2.5-pro	0.42	0.68	0.70	0.34	0.93	0.63	0.39	0.52	0.64	0.60	0.2	0.46
gemini-2.5-flash*	0.38	0.36	0.48	0.33	0.58	0.39	0.31	0.13	0.6	0.33	0.38	0.44
gemini-2.5-flash	0.17	0.35	0	0.24	0.07	0.02	0	0.25	0.42	0.66	0	0
gemini-2.0-flash	0	0.06	0.28	0.32	0.02	0.30	0	0.15	0	1	0	0
deepseek-r1	0.49	0.58	0.79	0.42	0.50	0.36	0.47	0.25	0.19	0.48	0.05	0.08
deepseek-v3	0.06	0.03	0.29	0.25	0.30	0.12	0.16	0.29	0.16	0	0	0
claude-4-sonnet*	0.24	0.28	0.40	0.55	0.58	0.43	0.31	0.21	0.38	0.13	0.22	
claude-4-sonnet	0.43	0.14	0.24	0.67	0.17	0.30	0.24	0.41	0	0.27	0	0.12
claude-3.7-sonnet*	0.38	0.46	0.47	0.53	0.30	0.26	0.41	0.29	0	0.3	0.09	0.44
claude-3.7-sonnet	0.35	0	0.73	0.88	0.02	0.52	0.04	0.33	0.08	0.14	0	0.3
claude-3.5-sonnet	0.22	0.17	1	0.83	0.06	0.03	0.19	0.29	0.13	0.25	0	0.41

3885

3886

3887

3888

3889

3890

3891

3892

3893

3894

3895

3896

3897

3898

3899

3900

3901

3902

3903

3904

3905

3906

Table 23: Detailed model performance on GSI task in the ORACLE benchmark 2@1.

Models	Easy Black-Box in GSI						Hard Black-Box in GSI					
	1	2	3	4	5	6	1	2	3	4	5	6
qwq-plus	0.45	0.61	0	0.26	0.13	0.11	0.43	0.34	0	0.33	0	0
qwen-plus*	0.64	0.70	0.85	0.56	0.69	0.32	0.38	0.68	0.43	0.80	0.33	0.46
qwen3-32b*	0.63	0.68	0	0.53	0.30	0.20	0.25	0.42	0	0	0.13	0.19
qwen3-235b*	0.44	0.36	0.68	0.43	0.14	0.26	0.47	0.68	0.08	0.90	0.05	0.12
o4-mini	0.72	0.76	0.77	0.76	0.21	0.10	0.37	0.31	0.21	0.27	0	0.23
o3-mini	0.5	0.62	0.88	0.59	0.15	0.22	0.49	1	0.13	0.26	0.06	0.12
o3	1	0.97	0.76	0.79	0.82	0.62	0.48	0.61	0.44	0.45	0.54	0.76
o1	0.18	0.68	0.89	1	0.05	0.20	0.58	0.74	0	0.29	0	0.44
gemini-2.5-pro	0.69	0.5	0.58	0.5	0.85	0.53	0.82	0.62	0.13	0.52	0.05	0.33
gemini-2.5-flash*	0.71	0.5	0.82	0.75	0.10	0.29	0.14	0.46	0	0.75	0.19	0.07
gemini-2.5-flash	0.24	0.81	0	0.57	0.09	0.18	0	0.15	0.06	0.71	0.15	0.10
gemini-2.0-flash	0	0.27	0.10	0.39	0.04	0.27	0	0.2	0.17	0.62	0	0
deepseek-r1	0.63	0.51	0.48	0.40	0.31	0.41	0.50	0.37	0.08	0.48	0	0.05
deepseek-v3	0.04	0.5	0.48	0	0.15	0.07	0.26	0.23	0.13	0.25	0	0
claude-4-sonnet*	0.88	0.78	0.27	0.81	0.56	0.48	0.61	0.45	0.25	0.12	0.13	0.62
claude-4-sonnet	0.49	0.17	0.55	0.78	0.36	0.30	0.18	0.28	0.3	0.26	0	0.2
claude-3.7-sonnet*	0.35	0.56	0.46	0.79	0.42	0.39	0.56	0.5	0.19	0.29	0	0.4
claude-3.7-sonnet	0.33	0.28	0.73	0.78	0	0.37	0.14	0.18	0.1	0.14	0	0.51
claude-3.5-sonnet	0.13	0.15	0.63	0.92	0.08	0.04	0.14	0.18	0	0	0	0.21

3925

3926

3927

3928

3929

3930

3931

3932

3933

3934

3935

3936

3937

3938

3939

3940

3941