GuessTheRuleBench: Evaluating LLM Agent Implicit Reasoning Capabilities with Concealed Rule Games (Benchmark Track) 4 Units Project Group Name: JAMAX The Dreamteam

Ali Shazal
University of California, Berkeley
3040810126

Michael Lu
University of California, Berkeley
3038358429

Xiang Zheng
University of California, Berkeley
3040874866

Juno Lee
University of California, Berkeley
3035642206

Arihant Choudhary
University of California, Berkeley
3035640399

Abstract

Abstract reasoning — the identification of patterns, comprehension of complex concepts implied both explicitly and implicitly, and the resolution of novel problems — is a critical measure of Large Language Model (LLM) agents' performance. However, existing benchmarks often fall short in providing dynamic and continuously challenging tasks, leading to evaluations that may favor memorization over genuine reasoning. We propose a novel, perpetually reusable benchmark based on "Guess-the-Rule" games, designed to dynamically generate new instances of tasks with randomly constructed rules, ensuring robustness against memorization. Our benchmark introduces four unique games, each with three difficulty levels, available through a Python library and a web application for LLM integration and human testing. Evaluation of four leading language models revealed significant variability in performance. Claude 3 Haiku and GPT-4o achieved the highest average win rates of 62.3% and 50.1%, respectively, across the Static Picnic, Dynamic Picnic, and Math Game. However, all models struggled with the Code Functions Picnic, failing to win any games. The overall average win rate of 40.7% and the decline in performance with increasing difficulty highlight the challenges LLMs face in implicit reasoning tasks and their limitations in concealed rule-based reasoning. As part our submission, we are sharing the public links to this project's presentation video, slides, code, and the static picnic dataset.

1 Introduction

The capabilities of LLMs have advanced significantly, enabling their application in tasks that demand complex reasoning and agentic behavior [8, 13]. These advancements have opened opportunities for LLMs to excel in tasks requiring dynamic adaptation and understanding of implicit rules. However, reasoning tasks that focus on implicit rule deduction remain underexplored compared to standard benchmarks, which primarily evaluate explicit reasoning capabilities [6, 10, 14]. Tasks requiring implicit rule induction, where models must infer and validate underlying patterns, are crucial for testing the adaptability and generalization of LLMs [10, 14]. Current benchmarks often fail to comprehensively evaluate such capabilities. Many existing benchmarks, including GSM8k [4], MATH [7], and HumanEval [3], have provided critical baselines for assessing problem-solving and reasoning. However, these benchmarks suffer from limitations such as dataset contamination and overfitting, which can undermine their utility for abstract reasoning evaluations [6]. Efforts to address these issues, such as removing contaminated partitions, manually constructing fresh benchmarks, or restricting test data access, have provided only temporary fixes [6]. Dynamic benchmarking frameworks that can generate novel problems on demand have been proposed as a more robust alternative but remain underexplored for abstract reasoning [5, 6].

To address this gap, we propose GuessTheRuleBench, a dynamic framework of "guess-the-rule" games designed to evaluate the implicit rule deduction capabilities of LLMs. Unlike prior benchmarks, GuessTheRuleBench focuses on iterative reasoning and dynamic rule induction, challenging models to operate in environments where rules are not explicitly provided [10]. By emphasizing implicit reasoning, this benchmark complements existing approaches and offers a unique lens through which to assess the reasoning capabilities of LLMs.

Specifically, we contribute the following:

1. An open-source benchmark featuring four unique games: Static Picnic, Dynamic Picnic, Code Functions Picnic, and Math Game.
2. Three progressively challenging difficulty levels, i.e. L1, L2, and L3, for each game, designed to increase complexity by expanding the dimensions of the underlying rules.
3. A public Python library for programmatically accessing and playing the games, to enable integration with LLM agents and workflows.
4. A web application where human users can play the games, or trigger real-time gameplay of OpenAI models engaging with the benchmark. This is for demo and documentation purposes.
5. A comprehensive set of metrics for evaluating player performance, available both in the Python library and on the web app.

2 Related Work

Existing benchmarks for evaluating the reasoning capabilities of LLMs have been instrumental in advancing the field. Benchmarks like LiveBench [13] focus on assessing execution and planning tasks, enabling researchers to measure the adaptability and performance of LLMs on structured datasets. However, these benchmarks primarily evaluate explicit reasoning capabilities, leaving implicit reasoning tasks underexplored [14].

GSM8k [4], MATH [7], and HumanEval [3] are widely used benchmarks that have provided critical baselines for assessing problem-solving and reasoning capabilities. However, they exhibit significant limitations, such as dataset contamination and overfitting, which undermine their ability to evaluate abstract reasoning [6]. Attempts to address these issues, such as restricting test data access or constructing fresh benchmarks, have often been temporary solutions. These efforts underscore the need for dynamic benchmarking frameworks capable of generating novel problems [6].

LogicGame [10] introduces rule-based reasoning games to evaluate LLMs' execution and planning abilities, leveraging deterministic and verifiable steps to assess adherence to explicit rules. However, it focuses primarily on rule-following rather than rule induction, limiting its ability to evaluate LLMs' capacity for dynamically deducing implicit patterns. GameBench [6] evaluates strategic reasoning in multi-agent environments, using a variety of games to test decision-making and planning in competitive scenarios. While it provides insights into LLMs' strengths and weaknesses in strategic reasoning, its reliance on predefined game rules and scenarios restricts its applicability to tasks requiring implicit rule deduction. ImplicitAVE [14], by contrast, targets implicit reasoning through a multimodal dataset for attribute value extraction, combining contextual clues, images, and prior knowledge to infer implicit attributes. This approach underscores the challenges LLMs face in reasoning beyond explicit textual data and offers valuable insights for addressing implicit reasoning tasks through multi-modal methodologies.

GuessTheRuleBench builds upon these prior works by introducing a framework specifically designed to evaluate implicit reasoning capabilities. By requiring models to infer, validate, and apply rules dynamically, GuessTheRule addresses a critical gap in existing benchmarks. Additionally, the inclusion of open-source tools and real-time simulation capabilities ensures that this benchmark is accessible and extensible for the research community.

3 Benchmark Design

3.1 Overview

In GuessTheRuleBench, we present four guess-the-rule games as benchmarks to evaluate the reasoning capabilities of LLMs through implicit rule deduction tasks: Static Picnic, Dynamic Picnic, Code Functions Picnic, and Math Game. Each game presents a concealed rule that the player must deduce from examples provided by a game master. The games vary in the nature of their rules: linguistic, logical, programmatic, and numeric respectively. Each game is based on static or dynamic data sources. Examples of each game are presented in Table 1. To facilitate broad accessibility, GuessTheRuleBench is provided as a Python library (to be published on PyPi [2]). Researchers and engineers can integrate the benchmark into their workflows and agents. We also present a web application for demo and documentation where users can either play our guess-the-rule games themselves or trigger a real-time interaction of OpenAI LLMs [12] playing those games, with more models to be added to the app in the future. Below we explain the details of the benchmark, Python library, and web app. Throughout this paper, we use the term 'users' to refer to any entity using our framework, be it LLM Agents or humans.

3.2 Game Definition

In our benchmark, we define each guess-the-rule game as a two-player game with a game master who sets the rule and a player who attempts to guess it. The game master provides initial examples to the player to start the game, and in each subsequent turn the player can either provide their guess for the underlying rule or ask for more examples. The game ends when any of the following conditions are met: the game master runs out of examples, the maximum number of turns are exhausted, or the player guesses the rule correctly. We define three difficulty levels, namely L1, L2, and L3, with increasing difficulty going from L1 towards L3. The details of each difficulty level for each of the four games are provided in Section 4.

3.3 Data Sources

Our games can be split into two types based on the data sources: static and dynamic.

Static Games. They are based off of pre-downloaded, fixed data sources. Their rules and examples are already determined, vetted, and saved in the system. When a user requests a new game, the system returns one from the saved list. All examples presented in those games are based off of the saved entities. The 'Static Picnic Game' in Section 4.1 is an example of this game type. We pre-process and clean the data beforehand to match the format and requirements defined by the game itself. Details of the pre-processing for the static picnic game can also be found in Section 4.1.

Dynamic Games. They leverage LLMs to generate rules and corresponding examples in real-time. Unlike static games, which rely on pre-determined datasets, dynamic games create unique and novel instances for each game session, ensuring that the benchmark remains perpetually reusable and resistant to memorization by LLMs. The rule generation process begins with categorizing the rule space into a predefined taxonomy encompassing various rule types, such as attribute-based, categorical, logical, relational, and semantic. For each game instance, the system prompts the LLM with a description of the selected rule type and examples of similar rules. The LLM then generates a new rule adhering to these constraints. To maintain the quality and solvability of the rules, for the dynamic picnic game, a validation layer is implemented to filter out ambiguous or overly complex rules. Once validated, the rules are stored in a database and used to generate examples on-demand during gameplay. For the Code Functions Game and Math Game, we utilize code as the latent space to ensure reliability in rule generation. The game master first generates the underlying code, which is then executed to mitigate the uncertainty associated with the LLM's next-token prediction. If the generated code cannot be executed correctly, an error is returned, and the game instance is regenerated to maintain consistency and accuracy. This approach ensures that each game instance presents a fresh and challenging task for the player, be it an LLM agent or a human participant.

3.4 Game Play

Request A Game Instance. The user first requests an instance of the game by providing the following parameters:

- Game name from the list: static_picnic, dynamic_picnic, functions_picnic, or math.
- Difficulty level from the list: L1, L2, or L3.
- Number of initial examples. Users can start with a minimum of 1 example or as many examples as are available. The system will return an error if the number of requested examples exceeds the number of examples available.

Depending on the user parameters, an instance of the game is returned as an object with a unique UUID. Since the game begins with at least one example, this counts as the first turn of the game.

Request More Examples Once the game has initiated, the user can request more examples by specifying the number of examples desired (N). The system will return an error if the number of requested examples exceeds

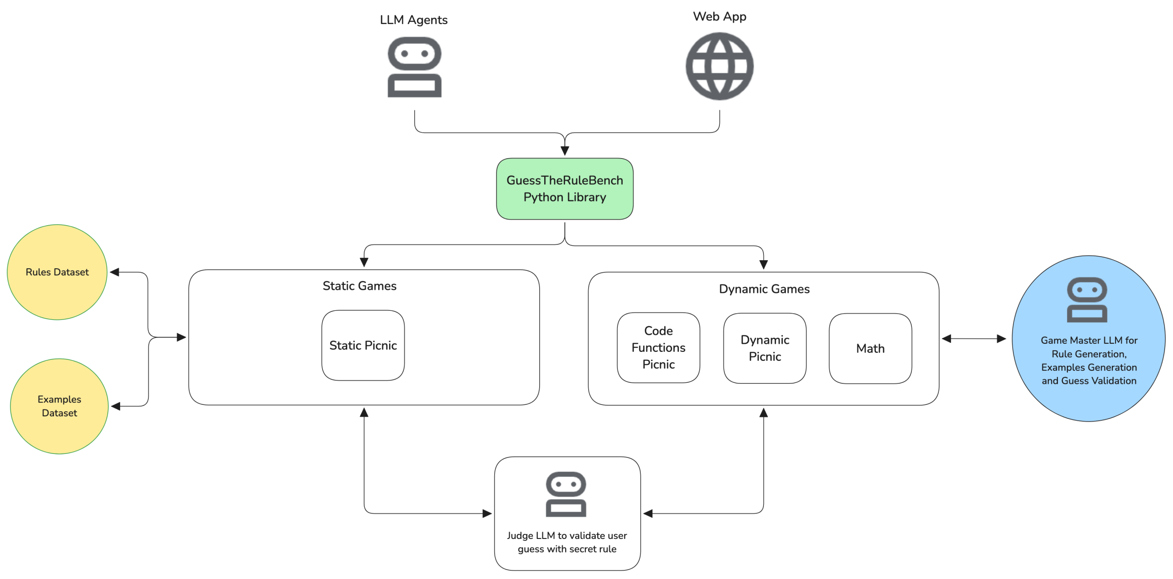

LLM Agents
Web App
GuessTheRuleBench Python Library
Rules Dataset
Examples Dataset
Static Games
Static Picnic
Dynamic Games
Code Functions Picnic
Dynamic Picnic
Math
Game Master LLM for Rule Generation, Examples Generation and Guess Validation
Judge LLM to validate user guess with secret rule

Figure 1: Game play interface on the web application

the number available. Each request for more examples counts as one turn.

Validate a Guess At any point after the game has initiated, the user can present their guess of the concealed rule and validate it with the system. The system returns either True or False. Each request for validating a guess counts as one turn.

Get Performance Summary The user can request a performance summary at any point in the game to view the number of turns taken, history of examples seen, and time elapsed. Checking the performance summary does not count as a turn.

User Strategy The user LLM agent is free to adopt any strategy depending on which metrics it aims to optimize. They can request many examples upfront, or try to guess early to minimize turns, or adopt hybrid strategies. The benchmark system provides the primitives needed to execute all these strategies.

3.5 Interface

We present two interfaces as part of the GuessTheRuleBench system: a Python library and a demo web app. A high-level design of the system can be seen in Figure 1.

3.5.1 Python Library

The Python library provides the four games as the following classes: *StaticGoingOnAPicnic()*, *DynamicGoingOnAPicnic()*, *CodeFunctionsPicnic()*, and *MathGuessTheRuleGame()*.

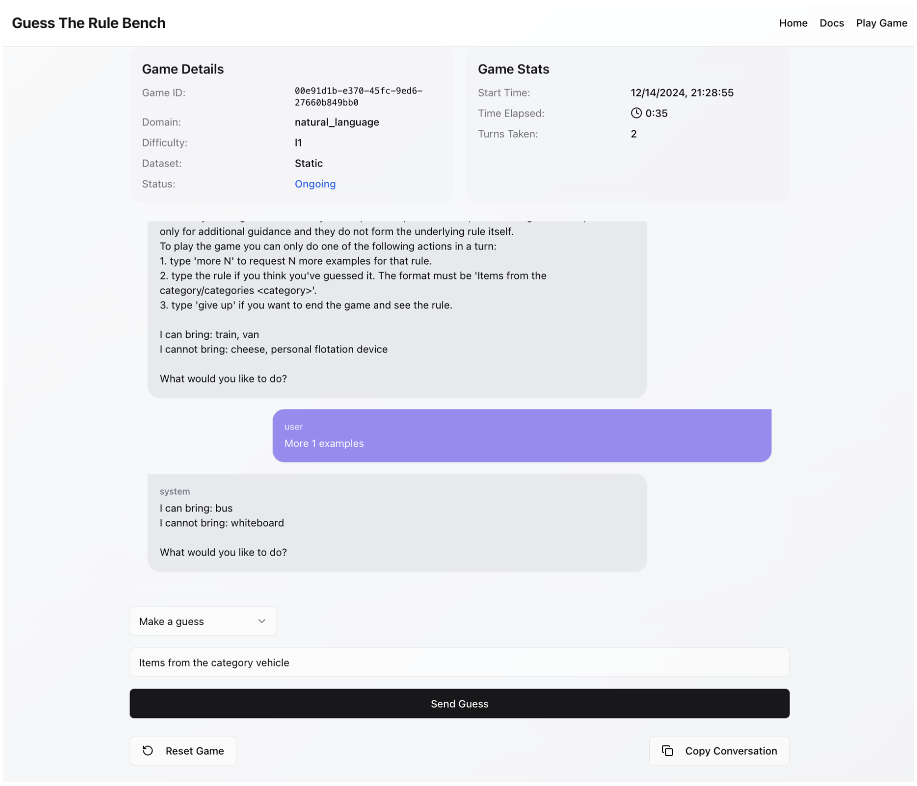
Each class has the following methods:

- *create_game_instance()* to request a new instance of the game.
- *get_more_examples(N)* to request N more examples.
- *validate_guess(guess)* to present the user's guess for validation.
- *get_game_summary()* to get the performance summary of the current game.
- *load_game(uuid)* to load a previously generated game.

3.5.2 Web Application

We have developed a demo web app that allows human users and researchers to interact with GuessTheRuleBench by playing the guess-the-rule games or observing LLM agents in real-time. The web app is designed to showcase the benchmark's capabilities and provide an intuitive interface for testing and analysis.

The *Play Game* page is the core of the web app, allowing users to directly interact with the guess-the-rule games.


Guess The Rule Bench

Game Details

Game ID: 00e91d1b-e370-45fc-9ed6-27660b849bb0

Domain: natural_language

Difficulty: l1

Dataset: Static

Status: Ongoing

Game Stats

Start Time: 12/14/2024, 21:28:55

Time Elapsed: 0:35

Turns Taken: 2

only for additional guidance and they do not form the underlying rule itself.
To play the game you can only do one of the following actions in a turn:
1. type 'more N' to request N more examples for that rule.
2. type the rule if you think you've guessed it. The format must be 'Items from the category/categories <category>'.
3. type 'give up' if you want to end the game and see the rule.

I can bring: train, van
I cannot bring: cheese, personal flotation device

What would you like to do?

user
More 1 examples

system
I can bring: bus
I cannot bring: whiteboard

What would you like to do?

Make a guess

Items from the category vehicle

Figure 2: Game play interface on the web application

Users choose one of the four games, set the difficulty level, select a player mode (either play the game themselves or observe an LLM playing it), specify the number of initial examples, and toggle between static and dynamic datasets.

The system provides examples for users to deduce the rule and allows interaction by making guesses, requesting additional examples, or ending the game. Performance metrics such as turns taken, examples seen, and total time are displayed in real-time at the top of the page. Figure 2 shows the gameplay page where the user is playing the game by themselves.

3.6 Evaluation Metrics

We use the following evaluation metrics for each game:

1. Turns Taken: Number of turns taken to complete the game starting with 1.
2. Duration: Time taken in seconds, beginning as soon as the first example is presented.
3. Number of examples seen by the user, starting with a minimum of 1.

The best performing player would take the fewest turns, see the fewest examples, and respond in the least amount of time.

4 Methodology

4.1 Static Picnic Game

The "Going on a Picnic" game is a reasoning task where players deduce an implicit rule based on examples provided by a host. The game involves distinguishing between items that can or cannot be "brought on a picnic," encouraging participants to infer abstract patterns and test hypotheses iteratively. This framework is relevant for evaluating the implicit reasoning capabilities of LLMs, as it challenges models to generalize and adapt to new scenarios by uncovering unseen patterns—key aspects of intelligent behavior.

In our Static Picnic Game, we utilize the Google Open Images Dataset [9] as the primary data source. The dataset's rich image labels and hierarchical organization of items serve as the foundation for our rule generation.

Game	Difficulty Level	Examples	Underlying Rule
Static Picnic	L1	Cream, toothbrush, diaper, sunscreen	Personal care items.
Dynamic Picnic	L3	Brexit, flint water crisis, Edward Snowden, moon landing	Items that have been mentioned in a Pulitzer Prize-winning newspaper article.
Code Functions Picnic	L1	Choenix, tucktoo, rudely, curney, sprong	Code: `return len(word) > 4 and word[0].lower() not in 'aeiou'` Natural Language: words with length greater than four and start with a consonant.
Math Game	L3	-1, -2, 1, 2, 5, 10, 13, 26, 29, 58, 61, 122, 125	The sequence multiply by 2 when index is even, and add 3 when index is odd

Table 1: Example of an instance of each of the four games with difficulty levels, true examples, and underlying rules.

Specifically, we use item categories (e.g., "personal care", "furniture", "mammal", etc) derived from the dataset to define rules such as "items belonging to category {category}." Difficulty levels are structured progressively: L1 rules involve a single category, L2 introduces combinations of two categories, and L3 includes rules spanning three categories, reflecting increasing levels of complexity. This setup results in 250 pre-generated rules and utilizes all 511 unique real-world items from the Open Images Dataset as the source of examples. By leveraging these hierarchical labels and predefined rules, the Static Picnic Game systematically evaluates the ability of LLMs to infer and generalize implicit rules under controlled, yet realistic, conditions. This approach offers a scalable and interpretable benchmark for assessing reasoning capabilities in large language models.

4.2 Dynamic Picnic Game

Distinguishing itself from the Static Picnic Game, which utilizes pre-defined rules and examples, the Dynamic Picnic Game generates rules and examples in real-time. This dynamic approach ensures a continuously challenging environment, effectively mitigating the risks of memorization, thereby providing a more authentic evaluation of reasoning capabilities.

Central to the Dynamic Picnic Game is a structured rule generation process grounded in a comprehensive taxonomy that categorizes rules into five distinct types: Attribute-Based, Categorical, Logical, Relational, and Semantic. This taxonomy ensures diversity and balance in rule creation. Rules are generated through tailored prompts directed at a Game Master LLM, specifying the rule type and providing illustrative examples. After that, examples are generated for each rule on the fly. When players request additional examples, the game master LLM generates new items consistent with the underlying rule. This process involves validating each example to ensure adherence to the rule, thereby maintaining the game's integrity and consistency.

To accommodate varying levels of challenge, the Dynamic Picnic Game incorporates all three difficulty levels: L1, L2, and L3. These levels are defined based on conceptual complexity and the degree of abstraction required. L1 difficulty features straightforward, attribute-based or categorical rules, while L2 level introduces logical or relational complexities. L3 level presents highly abstract and multi-layered rules, demanding sophisticated reasoning from players. This stratification ensures a progressive escalation in difficulty, enabling a comprehensive assessment of reasoning across different complexity levels. Addressing challenges such as rule ambiguity and example inconsistency, the Dynamic Picnic Game incorporates stringent validation criteria and real-time cross-checks. These measures prevent the generation of ambiguous or overly complex rules and ensure that all examples strictly adhere to the underlying rule. Additionally, to combat LLM hallucinations, a robust validation layer verifies all outputs from the LLM, enhancing the reliability of the Game Master's interactions and maintaining the overall integrity of the game.

Model	Static Picnic			Dynamic Picnic			Code Functions Picnic			Math Game		
	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3
GPT 4o	83.3	**90.0**	**80.0**	21.0	13.3	13.3	0.0	0.0	0.0	**100.0**	40.0	**60.0**
GPT 4o Mini	83.3	73.3	70.0	43.3	30.0	40.0	0.0	0.0	0.0	**100.0**	**80.0**	0.0
Claude 3.5 Haiku	83.3	80.0	76.7	30.3	23.3	16.7	0.0	0.0	0.0	**100.0**	60.0	30.0
Claude 3 Haiku	**86.7**	76.7	56.7	**90.0**	**80.0**	**73.3**	0.0	0.0	0.0	**100.0**	40.0	20.0

Table 2: Win rate of different models across games and difficulty levels. Bold values highlight the best performances in their respective columns.

4.3 Code Functions Picnic Game

The Lexical Function Game tasked LLM agent players with guessing programmatic rules—i.e., Python string manipulation code snippets. The code for these functions was dynamically generated by a Game Master LLM agent.

The use of LLM-generated code snippets presented a few problems to overcome. We used an agentic code execution loop to re-generate functions until our game master agent outputted valid Python syntax. Additionally, we noticed that many generated functions did not evenly split the domain of English words on True/False outputs—for example, the `is_valid_palindrome` function almost always returned False. Since the player almost always received negative examples, there was insufficient information to guess the rule. To combat this, we added a check that the generated function returned True or False on at least 10% of sampled dictionary words. Combined with retrieval augmentation on previously generated legal and illegal functions, we observed that the game master was able to generate diverse rule functions that were reasonably guessable, at least by a human player. The limitations of string-manipulation-based rules, combined with the poor performance of frontier LLMs on even simple rule functions, meant that we only included a single difficulty level. Otherwise, the structure of our code function game and its API was similar to the other game types.

4.4 Math Game

The Math Game is designed to test the models on their ability to deduce mathematical rules. It focuses on the ability to reason about numerical relationships and solve problems that involve mathematical patterns. The players are provided with a series of numerical examples that follow a specific mathematical rule. These examples may include operations like addition, subtraction, multiplication, division, and, for different indices, the sequence may obey different rules. This game also utilizes a Game Master LLM to generate rules. Due to the uncertainty in the next-token prediction ability of LLMs, we do not directly task the LLM with generating the sequence. Instead, we first prompt the game master to produce a Python function in a specific format, and after that, we ask it to execute the function to create examples.

The Math Game also incorporates all three levels of difficulty with different mathematical operators. L1 rules involve only basic operations such as addition, subtraction, multiplication, and division. The entire mathematical sequence follows a single, consistent rule. For L2, the generation process is similar to L1 but introduces negative values. This addition can result in abrupt changes (e.g., jumps) during multiplication and division. At L3, the rules vary based on the index of each value. This requires the LLMs to infer the hidden index-based rule (which is implicit) in addition to identifying patterns directly from the values.

5 Experiments and Results

We created agents with 4 leading language models and evaluated them on all 4 games in our benchmark. The chosen models were: GPT 4o, GPT 4o mini (variants of GPT-4 [12]), Claude 3.5 Haiku, and Claude 3 Haiku (variants of Anthropic's Claude [1]). We set up each agent using their official platform APIs and ran experiments on all three difficulty levels. We also included a *max_turns* parameter in each experiment to constrain the model to finish the game within a limit—this was based on our initial experiments where some instances would run for a large number of turns. Each combination of the game, difficulty level, and *max_turns* was run 5 times to account for the probabilistic nature of model outputs. The full results for all four games can be found in Section 7.1 in the appendix. The aggregated results are presented in Table 2.

5.1 Model Capabilities

We see that in Dynamic Picnic games, Claude 3 Haiku outperformed all models across all three difficulty levels with 90%, 80%, and 73.3% win rates in L1, L2, and L3 difficulty levels respectively. The other three models struggled significantly, with none of them being able to cross the 50% win rate threshold. This resulted in a substantial gap between the win rate of Claude 3 Haiku and all other models. It is also interesting to note that the bigger models (GPT 4o and Claude 3.5 Haiku) performed worse compared to the smaller models. **In this game, the average win rate across all models was only 39.5%.**

In the Static Picnic game, GPT 4o had the highest win rate on the L2 and L3 difficulty levels, with Claude 3 Haiku scoring slightly higher on L1 difficulty. **Generally, all models performed better in this version of picnic, with an average win rate of 78.3% across all models, an almost 2x increase from the dynamic version of the game.**

The Math game saw the GPT 4o models outperforming the Claude models. While all models achieved a 100% win rate on the L1 difficulty, GPT 4o Mini successfully won 80% of the L2 games, and GPT 4o won 60% of the L3 games. As we observed in the other games, there is a downward trend in the win rate with increasing difficulty levels. The overall average win rate of 40.7% of all models and the decline in performance with increasing difficulty levels highlight the challenges LLMs face in implicit reasoning tasks and their limitations in concealed rule-based reasoning.

5.2 Failure in Code Function Game Results

In the Code Function Picnic game, every model was completely unable to guess even the simplest string manipulation rules. This is a marked contrast to the surprisingly competitive performance of most LLM agents when tasked with guessing meaning-based rules. Notably, the LLMs being benchmarked displayed an inability to self-correct or reason about what kind of rule would be more likely (e.g., vowel count, word length) given a set of examples. Instead, the LLM players tended to hallucinate unrelated rules. When prompted for chain-of-thought, models would similarly hallucinate obviously false facts about example words, such as miscounting letter counts or making incoherent claims.

No amount of prompt optimization substantially improved this performance. Even when GPT 4o was interactively guided by a human in real time (through OpenAI's ChatGPT web interface [11]) who pointed out incorrect claims, the model was ultimately unable to guess the rule without being explicitly told the answer. We therefore conclude that, while LLM agents can reason strongly about the meaning and context of words, they still struggle immensely with meta-level, out-of-domain tasks such as string manipulation.

6 Conclusion and Future Work

In this paper, we introduced GuessTheRuleBench, a novel benchmark designed to evaluate the implicit reasoning capabilities of LLMs through a series of "guess-the-rule" games. Our results highlight both the potential and the limitations of existing LLMs in the domain of implicit rule deduction. Notably, models like Claude 3 Haiku and GPT-4o demonstrated strong performance in the Static Picnic, Dynamic Picnic, and Math games, with win rates as high as 62.3% and 50.1%, respectively. However, none of the models succeeded in the Code Functions Picnic game, underscoring significant challenges LLMs face in reasoning about programmatic rules. The noticeable decline in win rates as game difficulty increased further illustrates the need for more advanced methods to improve LLMs' generalization and adaptability in rule-based reasoning tasks.

Future work aims to address these challenges by incorporating more sophisticated multi-step reasoning tasks and exploring model fine-tuning or prompt engineering to better equip LLMs for the demands of programmatic reasoning. Additionally, the integration of human-in-the-loop feedback could provide LLMs with corrective guidance, thereby fostering more robust self-correction mechanisms. Expanding the benchmark with more diverse game types and dynamic rule generation methods will further stress-test LLM capabilities, offering the community a comprehensive and ever-evolving evaluation platform for implicit reasoning in LLMs.

References

[1] Claude: An ai assistant by anthropic. https://www.anthropic.com/index/claude. Accessed: 2024-12-19.

[2] Pypi - the python package index. https://pypi.org/. Accessed: 2024-12-19.

[3] Mark Chen, Jerry Tworek, and Heewoo et al. Jun. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

[4] Karl Cobbe, Vineet Kosaraju, and Mohammad et al. Bavarian. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

[5] Yong Fan and Mingyu Chen. Nphardeval: Dynamic benchmark generation for complex reasoning tasks. arXiv preprint arXiv:2401.01234, 2024.

[6] Maria Garcia and Ronak Patel. Gamebench: A multi-agent environment for evaluating strategic reasoning in llms. In *Advances in Neural Information Processing Systems*, 2023.

[7] Dan Hendrycks, Collin Burns, and S. et al. Kadavath. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874, 2021.

[8] Alice Jones and Bo Wang. On the use of benchmarks for evaluating large language models. *Journal of AI Research*, 75:1–20, 2024.

[9] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, and et al. Open images dataset. https://storage.googleapis.com/openimages/web/index.html. Accessed: 2024-12-19.

[10] Carol Lee and David Kim. Logicgame: Evaluating rule-based reasoning in language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023.

[11] OpenAI. Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt, 2022. Accessed: 2024-12-19.

[12] OpenAI. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[13] John Smith and Jane Doe. Livebench: Benchmarking llms in real-time execution scenarios. In *Proceedings of the 2023 Conference on Neural Information Processing Systems*, 2023.

[14] Wei Zhang and Xinyu Li. Implicitave: Multimodal attribute value extraction for implicit reasoning. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

7 Appendix

7.1 Experiment Results

7.1.1 Static Picnic

The Static Picnic game has a fixed dataset, so we include additional columns for the total examples available and the number of examples seen, as shown in Table 3. One thing to note here is that the ”total examples available” column only presents the sum of positive examples, because the negative examples are essentially every other data point in the dataset.

Model	Level Difficulty	Rule Guessed	Games Played	Avg Time (s)	Turns Taken	Positive Examples Seen	Negative Examples Seen	Total Examples Available
`Claude 3.5 Haiku`	L1	False	5	1.52	1.00	2.00	2.00	40.80
	L1	True	25	8.10	4.32	11.44	11.44	24.64
	L2	False	6	2.28	1.33	2.33	2.33	9.83
	L2	True	24	6.95	3.63	7.13	132.54	8.33
	L3	False	7	3.12	1.71	2.86	74.43	6.00
	L3	True	23	5.23	2.78	4.35	113.87	4.39
`Claude 3 Haiku`	L1	False	4	5.22	3.75	7.75	7.75	30.50
	L1	True	26	4.44	3.31	6.88	6.88	22.85
	L2	False	7	3.54	3.14	6.14	6.14	7.57
	L2	True	23	3.57	2.65	5.52	5.52	11.00
	L3	False	13	2.25	2.00	3.46	3.46	3.85
	L3	True	17	2.69	2.24	4.06	4.06	4.76
`GPT 4o`	L1	False	5	3.07	3.20	7.60	7.60	21.80
	L1	True	25	1.99	2.80	7.36	7.36	23.52
	L2	False	3	2.46	3.33	8.33	8.33	8.33
	L2	True	27	1.80	2.33	5.70	5.70	11.52
	L3	False	6	1.46	1.83	3.83	3.83	4.50
	L3	True	24	1.61	2.13	3.96	3.96	5.21
`GPT 4o mini`	L1	False	5	2.14	4.20	13.40	13.40	21.80
	L1	True	25	2.57	4.92	13.12	13.12	22.68
	L2	False	8	1.32	2.00	3.88	3.88	6.50
	L2	True	22	2.14	3.00	6.68	6.68	8.95
	L3	False	9	1.37	1.56	2.78	2.78	4.00
	L3	True	21	1.86	3.00	7.05	7.05	7.57

Table 3: Performance Metrics for *StaticGoingOnAPicnic* Game Across Models and Difficulty Levels

7.1.2 Dynamic Picnic

The summary of the Dynamic Picnic game experiment is shown in Table 4.

Model	Level Difficulty	Rule Guessed	Games Played	Wins	Losses	Avg Time (s)	Turns Taken
`Claude 3.5 Haiku`	L1	False	20	0	20	98.40	6.65
	L1	True	10	10	0	40.46	6.80
	L2	False	23	0	23	85.48	5.74
	L2	True	7	7	0	134.32	9.71
	L3	False	25	0	25	90.16	6.12
	L3	True	5	5	0	140.94	10.40
`Claude 3 Haiku`	L1	False	3	0	3	24.21	4.00
	L1	True	27	27	0	34.19	4.70
	L2	False	6	0	6	62.77	4.33
	L2	True	24	24	0	46.91	4.96
	L3	False	8	0	8	57.51	6.63
	L3	True	22	22	0	45.54	3.86
`GPT 4o`	L1	False	26	0	26	16.32	6.42
	L1	True	7	7	0	13.76	5.86
	L2	False	26	0	26	19.19	6.81
	L2	True	4	4	0	24.47	7.00
	L3	False	26	0	26	22.93	6.69
	L3	True	4	4	0	21.49	7.75
`GPT 4o mini`	L1	False	17	0	17	11.96	5.82
	L1	True	13	13	0	16.09	7.62
	L2	False	21	0	21	12.34	5.86
	L2	True	9	9	0	16.29	7.67
	L3	False	18	0	18	20.36	4.89
	L3	True	12	12	0	26.42	9.58

Table 4: Performance Metrics for *DynamicGoingOnAPicnic* Game Across Models and Difficulty Levels

7.1.3 Code Functions Picnic

None of the models won any Code Functions Picnic game, as shown in Table 5.

Model	Level Difficulty	Rule Guessed	Games Played	Wins	Losses	Avg Time (s)	Turns Taken	Examples Seen
Claude 3.5 Haiku	L1	False	20	0	20	10.84	4.00	10.0
Claude 3 Haiku	L1	False	20	0	20	14.94	3.60	9.2
GPT 4o	L1	False	20	0	20	74.41	3.70	9.4
GPT 4o mini	L1	False	20	0	20	52.36	3.55	9.1

Table 5: Performance Metrics for *CodeFunctionPicnic* Game Across Models at Difficulty Level L1

7.1.4 Math Game

The summary of Math Game experiments is shown in Table 6.

Model	Difficulty	Result	Games Played	Avg. Time (s)	Avg. Turns Taken	% Turns Taken
`claude-3-haiku`	L1	Win	5	4.67	2.8	28.00
	L2	Loss	3	32.74	15.0	100.00
	L2	Win	2	7.90	5.5	36.67
	L3	Loss	4	51.82	20.0	100.00
	L3	Win	1	122.25	16.0	80.00
`gpt-4o`	L1	Win	5	1.64	1.6	16.00
	L2	Loss	3	26.47	15.0	100.00
	L2	Win	2	3.02	2.0	13.33
	L3	Loss	2	39.02	20.0	100.00
	L3	Win	3	11.05	7.0	35.00
`gpt-4o-mini`	L1	Win	5	10.10	1.6	16.00
	L2	Loss	1	50.78	15.0	100.00
	L2	Win	4	5.26	1.5	10.00
	L3	Loss	5	48.18	20.0	100.00

Table 6: Performance Metrics for the *Math Game* Across Models and Difficulty Levels.

7.2 Prompts

7.2.1 Static Picnic

The Static Picnic game uses GPT 4o mini as the judge for player's guess validation. The prompt is shown below.

```
Determine if the following user guess is semantically equivalent or reasonably close in meaning to the actual rule. Consider synonyms, related terms, and general concepts. If the user's answer is correct according to the examples but deviates slightly from the rule, it can still be marked as correct.

User Guess: {guess}
Actual Rule: {rule}
Examples Shown: {positive_examples}

Respond with 'yes' if they are equivalent or similar, otherwise respond with 'no'.
```

7.2.2 Dynamic Picnic

Rule Generator Prompt:

```
You are the game master for ”Going on a Picnic.” You have already created the following rules and should not
repeat them: {existing_rules_text}

Your task is to create a unique and creative attribute-based rule that determines which items can be brought to
the picnic.

- The rule should be based on a specific attribute of the items (e.g., items that contain double letters, items with
exactly five letters).
- Avoid common attributes like color or starting letter.
d - Think of less obvious attributes to make the game challenging.
- Clearly state the rule.

Do not include any examples or additional text.

Format:
Rule: [Your unique attribute-based rule]
```

Rule Difficulty Classifier Prompt:

```
You are an expert in analyzing ”Guess the Rule Games.” Your task is to classify a given rule into one of three
complexity levels based on how abstract and conceptually challenging it is:

- L1: The rule is straightforward and concrete, requiring little to no abstraction.
- L2: The rule is moderately abstract or involves some logical reasoning.
- L3: The rule is highly abstract, conceptually difficult, or relies on complex, non-trivial reasoning.

Consider the complexity and level of abstraction required to comprehend the given rule. Then assign it one of
the labels: L1, L2, or L3.

Rule: {rule}

Respond with only one label: L1, L2, or L3.
```

Examples Generator Prompt:

```
You are an assistant helping to generate examples for ”Guess the Rule Games”.

The secret rule is: {rule}

Here are the examples that have already been provided by the game master: {existing_generated}.
Here are the examples that the player has already guessed: {existing_guesses}.
Your task is to provide {num_examples} new and unique examples of items that satisfy the secret rule.
- Only provide the items in a simple, comma-separated list.
- Do not mention the secret rule.
- Do not repeat any examples already provided by the game master or guessed by the player.
- Do not provide any additional explanation or text.
Format: [item1], [item2], ..., [item{num_examples}]
```

Validation Judge Prompt:

You are an expert at identifying semantic equivalency in natural language. Your task is to evaluate whether a proposed rule guess semantically matches a predefined rule. Consider variations in phrasing, synonyms, or minor rewordings as valid matches, but disregard guesses that significantly deviate from the intended meaning. Provide a precise and accurate evaluation.

The predefined rule is: {rule}.
The player's rule guess is: {rule_guess}.

Determine if the player's guess semantically matches the predefined rule. Respond with either "yes" if the guess matches, or "no" if it does not. Do not provide any additional explanation or commentary.

Format:
Your final answer: yes or no

7.2.3 Code Functions Picnic

Rule Generator Prompt:

Output exactly one lambda function which outputs "True" or "False" according to some creative rule. The function should be aptly named. Then you should assign generated_fn to the function you create. This is executable code, so do not add any other uncommented explanatory text. Do NOT add grave markers for the code block. You should define a function using the "def" python keyword, and then assign generated_fn to that function in the locals dict. Do NOT use the "lambda" keyword–instead, use "def". Your function MUST match the signature in the examples (exactly ONE string input and a boolean output). Then have a line with "pass". Do NOT repeat any of the examples or rephrase any of their code–instead, create original functions.

Validation Judge Prompt:

Read the following function code, then read the user's guess. Determine whether or not the guess accurately describes the function. The user is playing a guessing game, where they receive examples along with True/False labels depending on what the rule function returns for each example. The user's guess must be more specific than just "it manipulates strings." They don't need to guess the exact implementation, but should accurately describe the behavior in simple English. Equivalent synonyms or slightly different phrasing is acceptable.

If the guess is accurate, respond with "YES." If not, respond with "NO."

7.2.4 Math Game

The Math Game uses GPT 4o mini as the rule generator and judge for player's guess validation. The prompt is shown below.

Rule Generator Prompt:

You are a helpful assistant that generates mathematical rules and Python functions.
L1 means for primary school students, L2 means for middle school students, L3 means for high school students. For different levels, the rule should differ.
For primary school students (L1) and middle school students (L2), the relation is between the adjacent values and can involve multiplication, subtraction, addition, or division of the previous value. For L2, you should multiply/divide a negative value.
For L1 and L2, the entire sequence should obey the same rule (no if statements).
For high school students (L3), for each index, the sequence will obey different patterns for odd and even indices. The rule should only use multiplication, subtraction, addition, or division operations and remain simple.
Provide the rule in a concise format and generate a Python function that implements this rule. The function input is the current value in the sequence and the index of the current value, and the output is the next value in the sequence.
Your response should include:
Mathematical rule: Your explanation of the rule in natural language
(Important) End your mathematical rule with a `$$` signal.
Example:
```
def generate_next(current_value, index):
    # Rule: Add 2 to the current value
    return current_value + 2
```
(Important) End your function with `&&`.
(Important) It's invalid if you first add 5 then subtract 3 or do other inverse procedures; in such a case, simplify your rule as "add 2".
Make sure the function is valid and can be executed to generate a sequence. The user will give you the difficulty of the game (L1, L2, L3) and you should generate the rule and corresponding function accordingly.

Validation Judge Prompt:

Evaluate whether a user's guessed rule accurately captures the underlying mathematical relationship of a given sequence generation rule. The guess must be precise about the exact mathematical operations and parameters involved.
If the guess fails to fully or correctly capture the relationship (e.g., stating a constant difference without specifying the exact difference value), it is incorrect. For example, the following guess is wrong: "My Guess is: Each sequence follows a linear pattern with a common difference, where the sequence can be expressed as $a_n = a + nd$." This is too vague without specifying the exact value of the difference.
Respond only with "True" or "False" based on whether the guess is correct or not.