Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2025)

# LLM Game Rule Understanding through Out-of-Distribution Fine-Tuning

**Bahar Bateni, Benjamin Pratt, Jim Whitehead**

University of California, Santa Cruz
bbateni@ucsc.edu, bepratt@ucsc.edu, ejw@ucsc.edu

## Abstract

Large Language Models (LLMs) have shown that a model pre-trained on general knowledge can perform well on specific tasks. However, LLMs natively perform poorly when it comes to demonstrating an understanding of rules, such as applying them, interacting with them, generating or modifying them, or evaluating them. Fine-tuning LLMs on a specific set of rules can significantly improve this performance. Yet, doing so undermines one of the main advantages of using a pre-trained model, which is its ability to generalize to rulesets outside its training distribution. This ability is critical for using LLMs as a tool in the game development process to give feedback or suggest rule modifications. In this paper, we introduce a framework for generating datasets to benchmark and train LLMs on their understanding of rules. We use Solitaire card games as our testbed for generating these datasets, as they have simple rules but offer a large space of possible variants, each played completely differently. We define a set of these variants using our custom Game Description Language (GDL) and use the framework to generate game progression questions, along with a textual explanation for each answer. Using these datasets, we conduct experiments to evaluate multiple LLMs on their understanding of rules, both with and without fine-tuning. Furthermore, we perform out-of-distribution evaluations in which the model is tested on rulesets it has not been trained on. Our results show that fine-tuning can improve the model's performance on both in-distribution and out-of-distribution rulesets, suggesting that training on rule-based datasets can improve general rule understanding of LLMs.

## Introduction

Large Language Models (LLMs) have shown that models pre-trained on broad, generalized knowledge can perform surprisingly well on a range of specific tasks, such as solving math problems, common sense reasoning, code generation, and more. However, without any specific training, they show poor performance when tested on their understanding of rules. This poor performance occurs both on game rules (Bateni, Pratt, and Whitehead 2025) or in a more general context, such as inferential rules (Wang et al. 2024) or instruction following based on logical rules (Zhou et al. 2025).



In games, rules govern the evolution of games state as it's influenced by the player or other objects (Grünvogel 2005). A game designer focuses on designing these rules and structures that results in an experience for the player (Tekinbas and Zimmerman 2003). For an automated system to assist in this process, for example through evaluation, modification or generation of these rules, it should show some understanding of the rules and how they affect the game.

As mentioned before, LLMs natively fail on showing this type of understanding. On the other hand, when explicitly trained on a specific set of rules, they can successfully show an understanding by interacting with the rules or applying them. For example, they have been shown to effectively play games or predict next state based on rules (Schultz et al. 2025). This performance can be improved when LLMs are paired with search algorithms such as Monte Carlo Tree Search (MCTS) or other Reinforcement Learning (RL) methods (Yao et al. 2023; Chu et al. 2025).

A key drawback of this rule-specific training is that by doing so, we fail to leverage a significant advantage of using LLMs, which is the ability to generalize across different rulesets. Not only is this generalization important for eliminating the need for post-training for every ruleset, but it can also enable LLMs to perform higher-level tasks such as creating, modifying, or assessing rules, which requires them to be able to generalize their knowledge to never-seen-before rules and rule combinations.

LLMs also fail to effectively understand game rules when a more generalized analysis of rules is required. In our prior work, we showed that LLMs struggle with analyzing rule synergies (Bateni, Pratt, and Whitehead 2025). They frequently make errors involving the event order, rule effect on a game state, rule interactions, and more. These findings suggest that fine-tuning on targeted datasets addressing each of these failures may improve this performance, as long as this fine-tuning can be generalized.

Addressing these points, this paper investigates how well LLMs can understand never-seen-before rulesets, and how much their performance can be improved with fine-tuning on other rule datasets. Specifically, we explore the following research questions:

1. How well can an out-of-the-box LLM understand game rules without any task-specific training?
2. To what extent fine-tuning on these specific set of rules

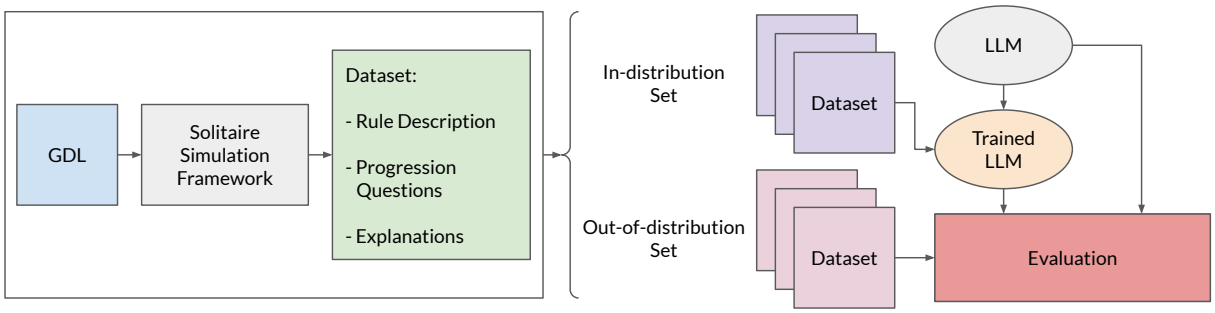


Figure 1: Overview of our system. We use a custom GDL to define various Solitaire games. For each game, our framework generates text-based game description, progression questions, and answer explanation. We split the generated datasets into in-distribution and out-of-distribution, where in-distribution is used for fine-tuning the LLM, and out-of-distribution is used for evaluation purposes. We compare the performance of LLM both with and without fine-tuning.

can improve performance? (in-distribution learning)

3. How well this fine-tuning generalizes to other games with new, unseen rulesets? (out-of-distribution learning)

To answer these questions, we develop benchmark datasets based on Solitaire card games. Solitaire games are simple single-player card games that combine a set of well-understood operational rules. The core gameplay always involves moving cards between different piles based on their ranks and suits. Despite this simplicity, they offer a vast space of possible combinations, making them a perfect testbed. Even though the rules are simple, it is still challenging for LLMs to reason about them. Moreover, because so many well-designed variants exist, we can evaluate how well LLMs can generalize across them.

We develop a framework for generating datasets of game progression questions across various Solitaire games. These questions ask whether a move is valid in a given game state and, if so, what the resulting state should be. Our system uses a custom Game Description Language (GDL) to define the rules of a Solitaire variant. For any set of rules, the system automatically generates a text-based description of the rules, a set of progression questions and answers including textual explanation of why a rule is valid or not. The texts are generated based on multiple templates and generalize to any set of rules describable by the GDL.

Using this framework, we evaluate several LLMs on these datasets, with and without fine-tuning, and measure their ability to generalize this new found understanding of rules to out-of-distribution (OOD) Solitaire variants. Figure 1 visualizes the overview of our evaluation pipeline on OOD datasets.

To summarize, our main contributions in this paper are:

- Introducing a framework to benchmark LLMs on rule understanding using Solitaire games.
- Analyzing the baseline ability of LLMs to reason about game rules and state transitions.
- Demonstrating how fine-tuning enhances rule understanding, even on previously unseen rule combinations.
- Evaluating the extent to which this improvement holds as the tested rulesets diverge further from those in fine-tuning.

Our findings indicate that fine-tuning LLMs on rule-based datasets can improve their general rule reasoning abilities.

## Related Work

### General Rule Understanding

Many studies have explored LLMs' ability to understand, follow, or apply rules. These rules can be as simple as single step instructions. For example, Mu et al. show LLMs struggle with following even simple instructions that specify or constraint model's behavior (Mu et al. 2024). Their findings show that the model fails to follow the rules, especially when the rules mandate a certain type of behavior instead of constraining it. Moreover, their experiments show supervised fine-tuning on rule-following tasks significantly improve the results. They conclude that fine-tuning can be a potential avenue for improving rule-following.

In addition to instructional rules, LLMs struggle to follow inferential rules. Sun et al. make a clear distinction between the two, and proposes RuleBench to evaluate LLMs on inferential rules (Sun et al. 2024). They highlight limitations of LLMs in following these rules, and categorize the errors into triggering errors, where the LLM fails to find which rule is triggered leading to incorrect results, and execution error, in which the rule is executed incorrectly. We observe similar error types in our experiments. Additionally, the authors show that fine-tuning on a synthetic dataset of rule-following can improve the LLMs rule-following capabilities. Their results suggest that these capabilities can be abstracted and learned from synthetic tasks and generalized to real-world scenarios, which aligns with our findings.

While Sun et al. focus on single-step reasoning with one golden rule and a few noise rules, Zhou et al. focuses on more realistic scenarios where many rules may affect the results (Zhou et al. 2025). They create RuleArena, a benchmark based on three practical domains: airline baggage fees, NBA transactions, and tax regulations. They show that these scenarios pose a significant challenge for LLMs, where they frequently struggle with choosing the correct rules or applying them. In more challenging scenarios, the accuracy rarely exceeds 10%, and the models often fail to recall all necessary rules.

## Game Rule Understanding

In the context of game rules, we categorize the existing work based on the type of rule understanding into three categories:

- Game Progression: Given the current game state and an action, the model should show understanding of how performing the action affects the state under the ruleset. In other words, it applies the rules on a specific scenario.
- Game Playing: Given the current state of the game, the model should show understanding of how to play the game under the ruleset by suggesting an action. Since the action is abstracted, the rules should be considered across all possible actions.
- Rule Space: The model should show understanding of the rule space, for example by generating, modifying, or evaluating the rules. Since both action and state are absent from the input, the rule space should be considered across all possible states and actions. Note that evaluating rules focuses on one ruleset whereas generating or modifying them focuses on many possible rulesets.

Both game playing and rule space types require the model to consider many possibilities, effectively performing of search. We see that many works utilize either an internal or external search to achieve this. We focus on the simpler task of game progression to eliminate this need and purely focus on how much the model can show an understanding of rules. All three categories require the model to understand the rules, although in many studies this understanding is specific to one or a few rulesets and cannot be generalized.

An example of rule understanding through game playing is explored by Schultz et al, where they leverage planning capabilities of LLMs in playing a set of boardgames, including chess, connect four, and hex (Schultz et al. 2025). They train a transformer model from scratch to generate the best moves and estimating move values for any given state from any of these games. In addition to game playing, their model also shows game progression by predicting the next state for any move it generates. They use this model both within an external MCTS process and by fine-tuning it to simulate an internal search process to achieve better performance. Importantly, while their proposed model generalizes its training set to novel states, they focus on planning aspects and never investigate generalization to out-of-distribution rulesets.

In an earlier work, Yao et al. enhance decision making in LLMs by introducing Tree of Thought (Yao et al. 2023). They evaluate this improvement by testing LLMs on a set of tasks, including playing Game of 24 and Mini Crosswords. Similarly, their focus is on enhancing decision-making in LLMs. Their work demonstrate another example of LLMs showing rule understanding both in the game playing and game progression categories, since the model both chooses the action to play and predicts the resulting state. However, while their model is not trained on a specific set of rules, generalized rule understanding is not explored. Furthermore, the rules in both Game of 24 and mini crosswords are fairly simple, and do not involve any rule combinations or variations of rules.

Chu et al. compare how different types of post-training, specifically fine-tuning and reinforcement learning, affect LLMs' game playing performance in General Points (Chu et al. 2025). General Points is a variant of Game of 24, where the starting numbers are represented with playing cards. Similar to our work, they also focus on how well LLMs can generalize to new rules when post-training is used. Their results suggest that fine-tuning degrades the model's performance on out-of-distribution rulesets, while reinforcement learning improves it. However, we only use progression questions, which eliminates the need to search the action space that RL algorithms excel at. Instead, our goal is to purely measure whether or not the rules are understood and applied correctly. Another main difference with our work is that rules in Solitaire variants have a greater impact on how the game is played, whereas the rule variations in General Points is limited to how card representations are interpreted: in one variation, the card suits are changed, which does not affect gameplay in any way, and in the other, the face cards represent different numbers.

Bateni and Whitehead measure LLMs ability to play a simplified version of *Slay the Spire*, where the cards themselves introduce new rules (Bateni and Whitehead 2024). Their results suggest that LLMs can play well in scenarios in which long-term planning is needed, whereas more traditional game playing methods such as lookahead agents excels where cards have immediate synergies. Their work is another example of LLMs showing rule understanding through game playing. Xia and Yang show that although LLMs cannot play this game as well as human players, their performance demonstrates a statically significant correlation with difficulty level perceived by human players (Xiao and Yang 2024).

Many works on LLMs interacting with or understanding game rules are focused on game playing capabilities of LLMs, and game progression is explored mainly in service of game playing. For rule space tasks, LLMs should not only understand the rules, but also be able to consider the rules over all possible states and actions. In our pervious work, we showed that LLMs struggle with this task when trying to analyze how rules affect each other (Bateni, Pratt, and Whitehead 2025). Our findings suggested that targeted fine-tuning on different datasets may improve this performance. Here, we investigate this improvement in game progression tasks and show fine-tuning can enhance LLMs' understanding of rules in both in-distribution and, more importantly, out-of-distribution games.

## Solitaire

Toward our goal of benchmarking LLMs on their rule understanding capabilities, we developed a framework to define and simulate Solitaire game variants. An existing framework for simulating Solitaire games, Solvitaire, is capable of simulating billions of game states in an exhaustive search process to determine the winnablity of any given game, via high levels of optimization.(Blake and Gent 2024) In contrast, our framework is mainly focused on providing a general definition for Solitaire variants with a high degree of freedom in defining rules through our custom GDL. We design our framework in a way to generate textual descriptions of the game rules, the conditions for each rule, and explana-

| | Rule Understanding Type | | | Training | OOD Rule Generalization |
|---|---|---|---|---|---|
| | Game Progression | Game Playing | Rule Space | | |
| (Yao et al. 2023) | ✓ | ✓ | × | × | × |
| (Schultz et al. 2025) | ✓ | ✓ | × | ✓ | × |
| (Chu et al. 2025) | ✓ | ✓ | × | ✓ | ✓ |
| (Bateni and Whitehead 2024) | × | ✓ | × | × | × |
| (Xiao and Yang 2024) | × | ✓ | × | × | × |
| (Bateni, Pratt, and Whitehead 2025) | × | × | ✓ | × | × |
| This paper | ✓ | × | × | ✓ | ✓ |

Table 1: Related works in different categories of game rule understanding.

tions of why or why not an action is legal within the rules. We communicate the game rules through natural language as opposed to GDLs or logical descriptions, both because natural language is more robust in generalization to other games, and because previous work have shown LLMs understand rules in natural language better (Sun et al. 2024).

## Solitaire Simulation Framework

We introduce our custom Solitaire Game Description Language (SGDL) and the accompanying framework for generating datasets for different Solitaire variants. As mentioned, we chose Solitaire as our testbed because of its diverse variants with significantly distinct gameplay despite the simple set of rules. In this section, we briefly describe SGDL and each component of the framework. Full documentation of this system is available on our github repository.[1]

### Overview

Solitaire games are single-player card games played with playing cards. Each game starts by shuffling the cards and positioning them on the board, often in piles. The cards can be either face-up or face-down. The gameplay generally involves moving cards between piles.

Our Solitaire simulation framework allows for defining the game in a custom GDL. For any game defined this way, the framework offers the following:

- Playable versions of the game via text-based terminal or a GUI created with PyGame (shown in figure 2), where the board elements and position of items are automatically generated based on the rules and the initial state
- General game playing bots for generating play traces and sampling random states and actions to create a game progression dataset
- A template-based generator capable of producing a textual description of the game and reasoning text for each sample in the dataset

[1]Our code, GDL and framwork documentation, full prompt, and all of the training and test datasets are available at https://github.com/iambb5445/SolitaireGDL


Figure 2: Screenshot of the Scorpion Solitaire variant in our Solitaire simulation GUI.

### Solitaire GDL

SGDL is designed with the goal of supporting a large set of Solitaire variants while keeping the language fairly simple. The SGDL file has the following schema:

- Variant Name
- `$cards`: This section defines the number of decks and describes which suits and cards are present in the game.
- `$initial`: This section defines the piles and initial positioning of the card. Draw piles can be of type `deal` (e.g. Spider family) or `rotate` (e.g. Klondike family). Any other pile can be defined with a custom name. For each pile, the number of cards initially in the pile should be defined. An tag can be used to define the initial facing of the cards (`FACE_ALL`, `FACE_LAST`, etc.). Optionally, the exact set of cards in the pile can be defined (e.g. Blind Alley's starts with the 4 Aces in the 4 Foundation piles). Note that piles with the same name always follow the same rules.
- `$actions`: This section defines all the possible actions in the game and the conditions for each of them to be legal. The actions can be either `move` which moves a single card, `move_stack` which moves a stack of cards, or `draw`. Note that not every condition is valid for every ac-

Listing 1: Example SGDL file `golf.sgdl`

```
Golf

$cards
DECK 1 {SPADES, HEARTS, CLUBS, DIAMONDS}

$initial
DRAW 16 DEAL FOUNDATION
FOUNDATION 1 FACE_ALL
COLUMN 5 FACE_ALL
COLUMN 5 FACE_ALL
COLUMN 5 FACE_ALL
COLUMN 5 FACE_ALL
COLUMN 5 FACE_ALL
COLUMN 5 FACE_ALL
COLUMN 5 FACE_ALL

$actions
MOVE COLUMN FOUNDATION
OR
    DESTSRC Rank ascending
    DESTSRC Rank descending
    AND
        DEST Rank K
        SRC Rank 1
    AND
        DEST Rank 1
        SRC Rank K

$win
PILE ALL COLUMN Empty
```

tion. For example, the `SRCSTACK` family of conditions are applied on the stack that is being moved, and can only be defined for `move_stack` actions.

- `$auto`: Optionally, a Solitaire game can have one or more auto-actions. These actions are continuously checked and performed if their conditions are met. For example, in Spider, a stack of 13 face-up cards from King to ace of the same suit that is positioned at the top of a pile automatically moves to the foundation pile as soon as it's created.
- `$win`: Finally, this section defines the win condition for the game.

An example of an SGDL file is shown in Listing 1. Golf is a Solitaire variant from the pairing family with similarity to TriPeaks. The game includes 1 foundation pile, 7 column piles, and 1 draw pile with 16 cards. The goal is to make all the columns empty by moving cards to the foundation. A card can be moved to the foundation if it's one higher or lower than the card currently at the top of the foundation pile. Kings and Aces are considered adjacent. Finally, the draw pile can be triggered to deal a card to the foundation, changing the top card.

## Conditions

As described earlier, the SGDL schema includes conditions for actions, auto-actions, and win states. Conditions can be composite condition trees that combines different subtrees by using And and Or operations. Alternatively, conditions can be plain conditions from one of the 4 categories below.

First, general conditions are conditions that are about the state of the game. For example, "all column piles should be empty" is the win condition for Golf, as shown in Listing 1. These conditions are valid as win conditions or as draw conditions (e.g. in spider, drawing is only possible if every column has at least 1 card).

Second, in `move` and `move_stack` actions and auto-actions, we define the source card as the card being moved (or the first card in the stack being moved), and we call the pile it belongs to the source pile. Destination pile is the pile that the source card is being moved to, and we call the last card of the destination pile the destination card. `SRC` and `DEST` conditions relate to source and destination card/pile and can only be defined for `move` or `move_stack` actions or auto-actions. These conditions can be about the rank or suit of these cards, or the size of the piles. Additionally, `DESTSRC` conditions can define some relation between destination and source cards, for example, having matching suits or ascending ranks.

Finally, specific to the `move_stack` actions and auto-actions, `SRCSTACK` conditions involve properties of the stack being moved, such as suit or ranks of the cards in the stack, or the size of the stack.

For full documentation of the conditions and their syntax, refer to our github repository.

## Text Generation

A key feature of our framework is the ability to generate text data needed to benchmark LLMs. After converting the GDL into a playable game, a text description of the game's rules and mechanics is generated. This includes the definition of the piles, actions and their conditions, auto-actions and their conditions (if applicable), and the win condition.

Additionally, given a state and a move, a text summary is generated that explains the conditions, whether or not each one is true, and a step-by-step breakdown of sub-conditions until the root condition is resolved. Examples of these summaries are included in our repository. Note that this explanation only details the condition for validating an action, not how the next state is generated. Instead, the instructions in the game description should be followed to create the next state, and we do not generate step-by-step reasoning on how to achieve this.

## Games

Our repository currently includes over 30 variants from Solitaire families of Spider, Klondike, Pairing and Free Cell. Although games in each family have similar goals and mechanic, they can be played very differently. In general, Spider family revolves around creating stacks of 13 cards of the same suit from King to Ace. Klondike family has the goal of moving all the cards one by one to the foundation piles. Pairing family revolves around matching cards with one of lower rank, higher rank, same rank, or ranks that add up to a specific number. Finally, games in the Free Cell family have Cell piles that can contain exactly one card. The exact ac-

| Spider | Klondike | Pairing | Free Cell |
|---|---|---|---|
| EasySpider | Blind Alley's | Doublets | Baker's Game |
| Friendly Spider | Double Klondike | Golf | Big Baker's Game |
| Scarab | Klondike | Thirteen | Big Free Cell |
| Scorpion | Legion | Vertical | Difficult Free Cell |
| Simple Simon | Thirty-Six | Wish | Easy Free Cell |
| Spider | Westcliff | | Eight Off |
| Spider (2 Suits) | Whitehead | | Free Cell |
| Spider (4 Suits) | | | |
| Wasp | | | |

Table 2: Some of the Solitaire variants and variant families available in our repository.

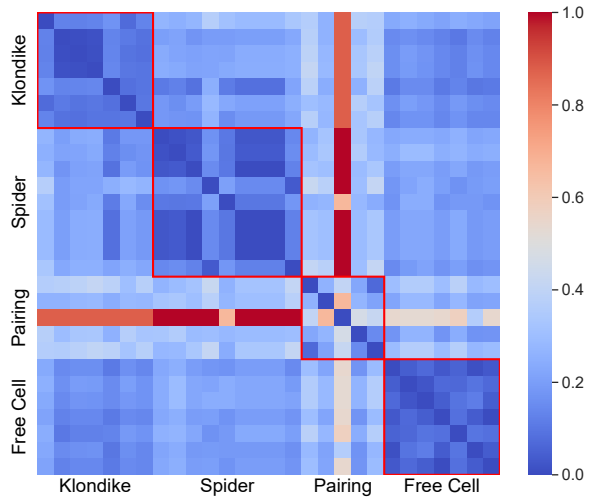


Figure 3: Distance matrix for Solitaire variants. Variants within the same family are closer, as expected.

tions and conditions differ in each variant. Table 2 lists some of these variations and their families.

Since variant definitions can vary across sources, we use *Free Solitaire* (WMGames 2024) as our sole reference for the rules, variant names, and family categorizations.

## Distance Metrics

The official family categorization helps us understand how similar different variants are. However, to better measure the differences, we develop a distance metric based on the number of rule differences.

We calculate the distance by counting the number of differences in mechanics, piles, and conditions. For each case, we consider differences in types and parameters. Furthermore, we ensure the metric is invariant to the order in which the conditions, actions, and piles are defined by considering all the possible permutations and using the one with the lowest distance. For example, changing a condition from "A and B" to "B and A" does not affect this distance.

For each rule, mechanic, and definition, we normalize the difference between 0 and 1 by dividing it by the number of comparison points. Finally, we average the distance over all rules, mechanics, and definitions. Figure 3 shows the distance between variants mentioned in Table 2. As we expect, the variants within the same family have a lower distance. The Pairing family is farther from the others since it focuses on matching ranks and does not care for suits. Thirteen is especially distinct from other variants outside its family, since it does not care about suits or ascending/descending ranks, and instead has a unique rule where ranks must add up to 13. We intentionally include Thirteen as one of our test datasets.

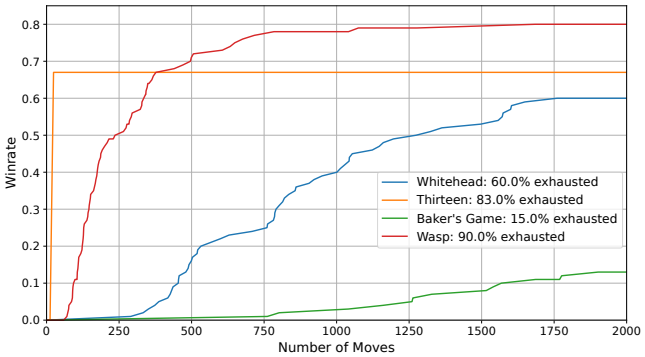


Figure 4: Winrates for a few variants using the DFS exhaustive agent. Exhaust percentage shows what percentage of the 100 games played have been fully explored after 2000 moves (some without reaching a win state, which happens when the initial state is unwinnable). Notice how rule variations result in gameplay diversity in terms of winrate or number of states.

## Simulation Agents

In order to generate datasets of game progression questions, we define general game playing agents that can choose an action given some state of any Solitaire variant. After an initial comparison between different game-playing methods, such as random agent and MCTS agent, we found that our exhaustive search DFS agent achieves higher win rates within the same time budget.

Given a game state, the DFS agent sorts all the possible actions based on a simple heuristic. The agent immediately wins the game if possible. Otherwise, the agent tries to avoid draw actions. Finally, the agent chooses the action with the highest number of follow-up actions to avoid dead-ends. The heuristic generalizes to any variant, since we do not use any variant-specific rules. The agent also avoids revisiting states and performs exhaustive search by backtracking when no new states are available.

Note that in some cases, the agent may fully exhaust all reachable state and never find a win state. However, we stop the agent after 2000 moves to avoid over-investing in one initial game state. Figure 4 shows the win percentage this agent achieves as it tries new moves.

We detail how this agent is used to generate training and tests datasets in the next section.

# Datasets

## Dataset Generation

To generate a dataset of $N$ game progression questions for variant $V$, we begin by randomly generating 100 initial states. These are generated by shuffling and dealing the cards into the initial board setup of $V$. Using the game-playing agent, we simulate gameplay for each of these 100 games until a win state or up to 2000 moves. Among all the unique states that the agent has reached in these 100 games either through making moves or backtracking, we sample $N$ states. The samples are spread across the 100 games with a near uniform distribution, except for significantly shorter traces, for which we under-sample.

For each of the $N$ sampled states, we randomly generate one possible action, with a 50-50 chance of being valid or invalid. Note that this action is completely random and is not the same as agent's original choice, as the agent's choice is biased by its heuristic and also can't be invalid.

This results in $N$ datapoints in the dataset, each containing a state and a random action for that state. Both state and action are represented by custom text formats. To eliminate imperfect information that requires model under evaluation to simulate probability, we include the value of face-down cards in the state representation. However, we use brackets to mark a face-down card, and the model is required to follow the rules around facing. For example, a face-down card can't be moved, and some actions may turn a card face-up.

Our Solitaire framework allows us to validate and execute any action on any state. We use this to add action validity and next state fields to each sample. The final dataset schema includes $N$ samples, each containing a game state, the proposed action, the step-by-step action legality explanation, a boolean indicating if the action is legal, and the next state.

We create 6 test datasets from 6 variants each with 500 samples. Sample sizes for the training datasets can be found in Table 3, and the variants included are detailed in Table 4. The following sections explains the rationale behind these choices.

## Training Datasets

We define 4 different variant sets for training. We create ft0 on only one variant and ft1 on 5 variants from the same family to see the effect of slightly diversifying training rulesets. We further test the effect of diversity by training ft2 on 5 games from 2 different families. Finally, we train ft3 on 10 games across 3 different families as our most varied dataset. Table 4 details the games included in fine-tuning each model.

To find a suitable sample size for training, we first compare the performance of ft0 models when trained on different sample sizes. Since we get the best result from 200 samples, we use this sample count for the rest of the training datasets. This sample count is spread across the included variants for each of the ft1, ft2 and ft3. Table 3 shows the exact sample counts per variant and total for each dataset.

All the fine-tuned models are based on GPT4o-mini. We use the smaller 4o-mini model because of the faster response, shorter fine-tuning time, and relaxed rate limits, which allowed for performing many experiments. Our goal in this paper is to analyze the effect of fine-tuning and the extent of generalization in LLMs as opposed to achieving the best possible performance.

| Model | #Sample | | #Games |
|---|---|---|---|
| | Per Game | Total | |
| ft0 (100) | 100 | 100 | 1 |
| ft0 (200) | 200 | 200 | 1 |
| ft0 (300) | 300 | 300 | 1 |
| ft1 | 40 | 200 | 5 |
| ft2 | 40 | 200 | 5 |
| ft3 | 20 | 200 | 10 |

Table 3: Sample count for training each fine-tuning model.

| Model | Datasets | | | |
|---|---|---|---|---|
| | Spider | Klondike | Pairing | Free Cell |
| ft0 | - | Blind Alley's | - | - |
| ft1 | - | Blind Alley's<br>Klondike<br>Westcliff<br>Thirty-Six<br>Whitehead | - | - |
| ft2 | Spider<br>Scorpion | Blind Alley's<br>Klondike<br>Thirty-Six | - | - |
| ft3 | Spider<br>Wasp<br>FriendlySpider<br>Spider (2 Suits) | Blind Alley's<br>Klondike<br>Thirty-Six | Golf<br>Vertical<br>Thirteen | - |

Table 4: Datasets used for fine-tuning each model.

## Test Datasets

We create 6 test datasets from 6 different variants each containing 500 samples. We carefully select these 6 variants based on their relations with the training sets for each model. These relations can be out-of-distribution (OOD), meaning samples from this game are not in the training set, or in-distribution (ID), meaning a seperate set of samples from this game are in the training set. We also consider in-family (IF), where the game is OOD for this model, but other games from the same family are included in its training set. Based on these, we choose the following variants for test datasets:

- Blind Alley's: in-distribution for all models
- Legion: in-family (but out-of-distribution) for all models
- Westcliff: in-distribution only for ft1
- Scorpion: in-distribution only for ft2
- Thirteen: in-distribution only for ft3
- Baker's Game: out-of-distribution for all models

## Prompt

Our prompt consists of several components. First, we explain our keywords, state and action text format, and variant

rules. Second, we define define the output JSON format that has the following schema:

- `"state"`: Reiteration the current state representation
- `"action"`: Reiteration of the current action
- `"thinking"`: Step-by-step explanation of action legality. For fine-tuning and few-shot examples, we use the summary generated by the framework for this property.
- `"legal"`: The final true/false value for action legality
- `"next_state"`: If the action is legal, this is the representation of the next state. Otherwise, this is `null`.

After the initial prompt, we inject 4 pre-generated requests and responses as our fewshot examples. We choose these examples randomly to ensure fairness, and we vary them sample to sample to avoid bias. Finally, we provide the current sample and ask the model to generate the requested JSON to answer the progression question.

## Experiments

We evaluate models on two accuracies based on percentage of correctly predicted values for the "legal" and "next_state" fields respectively. We consider next-state values only when the move is legal, since otherwise the next state value is undefined and correctly predicting it becomes a legality task, not a next state prediction task. Table 5 summarizes our results from the three experiments, which we discuss below.

### Native Performance

We first benchmark the out-of-the-box performance of LLMs on our test datasets to see how well they can perform the game progression tasks without any training. As shown in Table 5, GPT4o-mini performs poorly. The legal accuracy of this model averages on 0.67, while a random model would achieve an accuracy of about 0.5.

By examining the error cases, we find that the model makes errors in a two main ways. First, in the explanation summary generated by the model we see that the model may mention wrong rules or board information. For example, it can fail to include all of the rules, or mention the wrong rank or suit for a card. Second, it can execute the rules incorrectly. For example, the conditions are resolved to the wrong truth value, or applying And and Or operations is done incorrectly. These two error types match triggering errors and execution errors highlighted in RuleBench.

In addition to errors in predicting the legality of the rules, we see that the model can make mistakes in generating the next state by adding non-existent cards, omitting existing cards, or misplacing cards. On the other hand, the model does very well in reiterating input information, such as the current board state or the action in question.

For comparison, we also benchmark DeepSeek v3 and Llama 3.3 70B Instruct, which are significantly larger models.[2], and as expected, perform much better. However, the smaller GPT4o-mini model can reach and surpass their performance after fine-tuning in most cases.

[2]DeepSeek V3 and Llama 70B have 671B and 70B parameters respectively, while the number of parameters in GPT4o-mini is estimated at around 8B.

### Effect of Sample Count

In our second set of experiments, we analyze the effect of sample count. As mentioned, we fine-tune GPT4o-mini on samples from a single game (Blind Alley's) with different sample sizes. We find a 200 sample size to perform best in this scenario. Note that fine-tuning with any sample size improved the results in almost all cases, and we see a significant boost especially in predicting the next state.

### Effect of Fine-Tuning Distribution

In our final set of experiments, we compare models trained on datasets with different levels of diversity. For Blind Alley's which is in-distribution for all the models, we observe the best performance when fine-tuning is done exclusively on Blind Alley's (ft0). This is expected because the ft0 model not only has seen the highest number of samples from Blind Alley's, but also it has not been diluted by introducing other variants. In contrast, for Baker's Game, which is out-of-distribution for all the models, we get the best performance from ft3 which has seen the most variations.

For games that are only in-distribution for one of the models (Westcliff for ft1, Scorpion for ft2, and Thirteen for ft3), the best performance is achieved by the model that has the game in its distribution, which confirms our expectation.

We see that the generalization is more effective when combining similar rules in different ways than in responding to new rules. For example, when testing on Legion, ft1, which is trained on more games from Legion's family, does not perform better compared to ft2, which has the same number of games but from different families. Although Spider and Klondike families are different, they have similar rules ordered in different combinations. In contrast, Thirteen is a game where the rules completely differ from every other game, as shown in Figure 3. While ft3 performs slightly better on this game in predicting the next state due to the 20 samples it has from this game in its training set, we don't see much improvement in any of the other models compared to a model with no fine-tuning.

It's worth noting that the performance of ft3 on other variants may have suffered from inclusion of Thirteen in its training set, which is significantly different that other games.

Finally, we see in Figure 5, when the datasets are out-of-distribution for all models (Baker's Game and Legion), legal accuracy generally improves with more variations in the fine-tuning dataset. We attribute this to the generated step-by-step legality explanation, which helps the model to generalize. In comparison, next state accuracy doesn't always follow this pattern. Instead, it's more heavily influenced by whether the game is in-distribution, likely because we do not include a step-by-step explanation for it.

Based on our observations, we gain the following takeaways from this set of experiments:

- Fine-tuning is almost always beneficial, both for in-distribution and out-of-distribution rulesets.
- There is a limit to this generalization. If the out-of-distribution game differs significantly from the training (i.e. Thirteen), fine-tuning does not meaningfully improve the results.

| Model | FT | Test Dataset | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Klondike | | | | | | | | | Spider | | | Pairing | | | Free Cell | | |
| | | Blind Alley's | | | Legion | | | Westcliff | | | Scorpion | | | Thirteen | | | Baker's Game | | |
| | | L | NS | ID | L | NS | ID | L | NS | ID | L | NS | ID | L | NS | ID | L | NS | ID |
| GPT4o-mini | None | 0.63 | 0.14 | × | 0.62 | 0.14 | × | 0.65 | 0.11 | × | 0.55 | 0.03 | × | 0.81 | 0.00 | × | 0.75 | 0.29 | × |
| Llama 3.3 70B Instruct | None | 0.75 | 0.32 | × | 0.76 | 0.48 | × | 0.76 | 0.44 | × | 0.61 | 0.01 | × | 0.92 | 0.0 | × | 0.87 | 0.67 | × |
| DeepSeek-v3 | None | 0.86 | 0.55 | × | 0.80 | 0.63 | × | 0.82 | 0.66 | × | 0.67 | 0.29 | × | 0.89 | 0.02 | × | 0.93 | 0.87 | × |
| GPT4o-mini | ft0-100 | 0.85 | 0.73 | ✓ | 0.63 | 0.52 | ○ | 0.79 | 0.80 | ○ | 0.65 | 0.53 | × | 0.81 | 0.00 | × | 0.68 | 0.56 | × |
| GPT4o-mini | ft0-200 | 0.86 | 0.82 | ✓ | 0.69 | 0.67 | ○ | 0.77 | 0.83 | ○ | 0.72 | 0.56 | × | 0.83 | 0.00 | × | 0.73 | 0.79 | × |
| GPT4o-mini | ft0-300 | 0.82 | 0.73 | ✓ | 0.65 | 0.55 | ○ | 0.78 | 0.77 | ○ | 0.68 | 0.52 | × | 0.80 | 0.00 | × | 0.81 | 0.72 | × |
| GPT4o-mini | ft0-200 | 0.86 | 0.82 | ✓ | 0.69 | 0.67 | ○ | 0.77 | 0.83 | ○ | 0.72 | 0.56 | × | 0.83 | 0.00 | × | 0.73 | 0.79 | × |
| GPT4o-mini | ft1 | 0.84 | 0.67 | ✓ | 0.75 | 0.70 | ○ | 0.79 | 0.79 | ✓ | 0.66 | 0.39 | × | 0.76 | 0.00 | × | 0.76 | 0.72 | × |
| GPT4o-mini | ft2 | 0.86 | 0.77 | ✓ | 0.79 | 0.80 | ○ | 0.77 | 0.69 | ○ | 0.72 | 0.63 | ✓ | 0.79 | 0.00 | × | 0.76 | 0.70 | × |
| GPT4o-mini | ft3 | 0.78 | 0.59 | ✓ | 0.79 | 0.67 | ○ | 0.75 | 0.73 | ○ | 0.71 | 0.49 | ○ | 0.80 | 0.02 | ✓ | 0.88 | 0.86 | × |

Table 5: Results for each model. We use L for Legal Accuracy, NS for Next State Accuracy and ID for whether or not dataset is in-distribution. ✓: In-Distribution, ×: Out-of-Distribution, ○: In-Family but Out-of-Distribution.

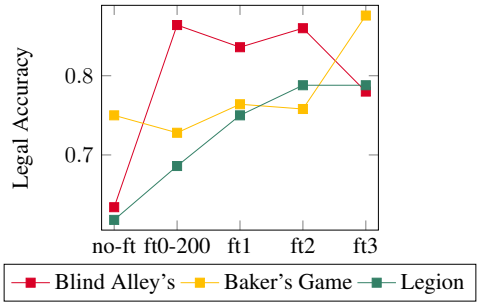


Figure 5: Accuracy of completely OOD or ID datasets.

- Fine-tuning on a broader set of games can make in-distribution performance worse since fewer samples of the same game are included, but it is generally beneficial for out-of-distribution performance. We see a sizable boost in OOD performance of ft3 (Baker's Game).
- Having a step-by-step explanation helps the model to better generalize to out-of-distribution samples as more variations are included in the fine-tuning dataset.

## Limitations and Future Work

One of our key contributions in this paper is our custom Solitaire simulation framework and its GDL. For keeping the GDL simple, we deliberately excluded a few Solitaire variants where the cards are not in piles, specifically, Pyramid and TriPeaks. Our goal is to expand this framework in the future to support these variants.

Additionally, we did not focus on optimizing the simulation performance or game playing bots. These features can speed up the dataset generation process, and possibly improve the quality of the sampled states. Furthermore, our sampling strategy can be improved by uniformly distributing the samples between move types or game stages.

In our datasets, we generate a text-based explanation for why a move is legal. However, we do not generate the same type of text for how the next state is generated. Doing so may improve the benefits we gain from fine-tuning.

Our work here focuses on the effect of fine-tuning on different in-distribution and out-of-distribution scenarios for a single model. Future work can extend this analysis to other, potentially stronger models. Moreover, while our main goal was to analyze the effect of fine-tuning and not to get the best results, it is important to note that the performance of fine-tuned models is still not satisfactory. There are several directions for improvement, such as integrating tool use for retrieving board information, exploring the use of an intermediate logical formalization for rules, or using reinforcement learning strategies to improve reasoning steps.

Finally, future work may explore generalized rule fine-tuning for improving rule understanding on problems such as performing and expanding game rules, game playing, or general rule-following. Furthermore, we believe that benchmarking on such datasets is important for directing the future generations of LLMs toward better rule understanding.

## Conclusion

In this paper, we introduced a system to benchmark the rule reasoning capabilities of Large Language Models based on Solitaire games. We presented a framework to simulate different Solitaire variants based on a custom GDL. Using these simulations, we created datasets from various solitaire families, and evaluated the performance of LLMs with and without fine-tuning.

Analyzing the results, we find that even a small number of finetuning samples can greatly improve model performance in most cases. Importantly, we see this improvement both on in-distribution and out-of-distribution rulesets, unless the ruleset substantially differs from the in-distribution set. Our findings show that including more diverse sample results in better generalization to out-of-distribution rulesets at the cost of reduced in-distribution performance.

## References


Bateni, B.; Pratt, B.; and Whitehead, J. 2025. Rule Synergy Analysis using LLMs: State of the Art and Implications. arXiv:2508.19484.

Bateni, B.; and Whitehead, J. 2024. Language-Driven Play: Large Language Models as Game-Playing Agents in Slay the Spire. In *Proceedings of the 19th International Conference on the Foundations of Digital Games*, FDG '24. New York, NY, USA: Association for Computing Machinery. ISBN 9798400709555.

Blake, C.; and Gent, I. P. 2024. The Winnability of Klondike Solitaire and Many Other Patience Games. arXiv:1906.12314.

Chu, T.; Zhai, Y.; Yang, J.; Tong, S.; Xie, S.; Schuurmans, D.; Le, Q. V.; Levine, S.; and Ma, Y. 2025. SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training. arXiv:2501.17161.

Grünvogel, S. M. 2005. Formal models and game design. *Game Studies*, 5(1): 1–9.

Mu, N.; Chen, S.; Wang, Z.; Chen, S.; Karamardian, D.; Aljeraisy, L.; Alomair, B.; Hendrycks, D.; and Wagner, D. 2024. Can LLMs Follow Simple Rules? arXiv:2311.04235.

Schultz, J.; Adamek, J.; Jusup, M.; Lanctot, M.; Kaisers, M.; Perrin, S.; Hennes, D.; Shar, J.; Lewis, C.; Ruoss, A.; Zahavy, T.; Veličković, P.; Prince, L.; Singh, S.; Malmi, E.; and Tomašev, N. 2025. Mastering Board Games by External and Internal Planning with Language Models. arXiv:2412.12119.

Sun, W.; Zhang, C.; Zhang, X.; Yu, X.; Huang, Z.; Chen, P.; Xu, H.; He, S.; Zhao, J.; and Liu, K. 2024. Beyond Instruction Following: Evaluating Inferential Rule Following of Large Language Models. arXiv:2407.08440.

Tekinbas, K. S.; and Zimmerman, E. 2003. *Rules of play: Game design fundamentals*. MIT press.

Wang, S.; Wei, Z.; Choi, Y.; and Ren, X. 2024. Can LLMs Reason with Rules? Logic Scaffolding for Stress-Testing and Improving LLMs. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7523–7543. Bangkok, Thailand: Association for Computational Linguistics.

WMGames. 2024. Free Solitaire.

Xiao, C.; and Yang, B. Z. 2024. LLMs May Not Be Human-Level Players, But They Can Be Testers: Measuring Game Difficulty with LLM Agents. arXiv:2410.02829.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 11809–11822. Curran Associates, Inc.

Zhou, R.; Hua, W.; Pan, L.; Cheng, S.; Wu, X.; Yu, E.; and Wang, W. Y. 2025. RuleArena: A Benchmark for LLM Rule-Guided Reasoning in Real-World Scenarios. In *Workshop on Reasoning and Planning for Large Language Models*.