Grammar and Gameplay-aligned RL for Game Description Generation with LLMs

Tsunehiko Tanaka, Edgar Simo-Serra

***Abstract*—Game Description Generation (GDG) is the task of generating a game description written in a Game Description Language (GDL) from natural language text. Previous studies have explored generation methods leveraging the contextual understanding capabilities of Large Language Models (LLMs); however, accurately reproducing the game features of the game descriptions remains a challenge. In this paper, we propose reinforcement learning-based fine-tuning of LLMs for GDG (RLGDG). Our training method simultaneously improves grammatical correctness and fidelity to game concepts by introducing both grammar rewards and concept rewards. Furthermore, we adopt a two-stage training strategy where Reinforcement Learning (RL) is applied following Supervised Fine-Tuning (SFT). Experimental results demonstrate that our proposed method significantly outperforms baseline methods using SFT alone. Our code is available at https://github.com/tsunehiko/rlgdg**

***Index Terms*—Large Language Model, Ludii, Game Description Language, Game Description Generation, Reinforcement Learning**

I. Introduction

Game Description Language (GDL) [1]–[5] is a domain-specific language used to represent a wide variety of games in a unified notation. For instance, the Ludii GDL [5] primarily models board games and covers more than 1,000 different game types. Game descriptions written in GDL are highly machine-readable, making it easy for dedicated game engines to run simulations. Because GDL descriptions can be automatically evaluated through simulation, they have become widely used in automated game design research [4], [6]–[8].

Recently, there has been increased interest in Game Description Generation (GDG) [9], [10]. GDG focuses on generating game descriptions from natural language texts, making it easier for non-experts to participate in game design. In GDG tasks, In-Context Learning (ICL) with Large Language Models (LLMs) [11] has shown great promise. LLMs excel in understanding textual contexts and can perform tasks with limited domain knowledge based on a few examples provided in prompts. For example, Hu *et al.* [9] demonstrated that enriching prompts with explanations of GDL notation and examples of game descriptions improves the accuracy of generated outputs. Additionally, Grammar-based Game Description Generation (GGDG) [10] proposed an iterative decoding method guided by GDL grammar rules, significantly enhancing grammatical correctness. However, the iterative improvements in GGDG have restricted grammatical accuracy and do not consider the actual gameplay behavior and features obtained through simulation.

The authors are with Waseda University, Tokyo, Japan. (Corresponding author: Tsunehiko Tanaka, email: tsunehiko@fuji.waseda.jp)

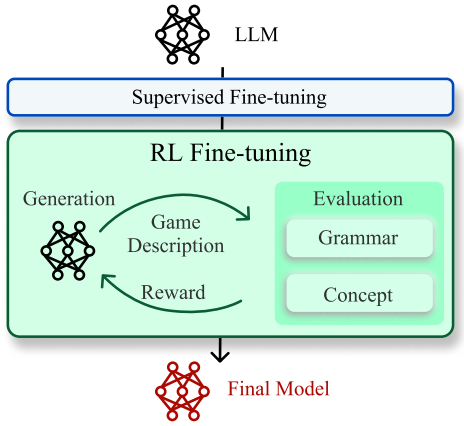

Fig. 1. **Overview of the proposed method.** The training process consists of two stages, starting with Supervised Fine-Tuning followed by Reinforcement Learning-based Fine -Tuning (RLFT), where rewards based on grammar and game concepts are utilized.

To address this issue, we propose Reinforcement Learning fine-tuning of LLMs for GDG (RLGDG), aiming to simultaneously enhance grammatical accuracy and fidelity of game features to the ground truth. Specifically, we design two types of rewards: (i) a grammatical reward evaluating whether the generated game description complies with GDL grammar, and (ii) a conceptual reward evaluating how accurately the generated game concepts [12], such as board cell usage and the proportion of states with multiple possible moves, align with the ground truth. As illustrated in Fig. 1, our proposed method employs a two-stage training procedure: first, Supervised Fine-Tuning (SFT) is conducted, which is then followed by Reinforcement Learning-based Fine-Tuning (RLFT) [13]–[15] based on reward optimization. By initially learning the basic grammar and structure of game descriptions through SFT, we mitigate unstable outputs in the early stages of RLFT. Experimental results demonstrate that our proposed framework outperforms baseline methods not only in grammatical correctness but also in the fidelity of game concepts.

Our main contributions are as follows:

- We propose Reinforcement Learning fine-tuning of LLMs for GDG (RLGDG) to jointly enhance grammatical accuracy and game feature fidelity.
- Our approach introduces grammatical and conceptual rewards to align generated game descriptions with grammar and actual gameplay features.
- We validate through extensive experiments that our proposed method significantly improves GDG performance.

II. Related Work

A. Game Description Language

Game Description Language (GDL) is a specialized language for describing specific games. In 2005, GGP-GDL [1] was introduced for General Game Playing, aiming to develop artificial intelligence agents capable of adapting to various games. Video Game Description Language (VGDL) [2] has been developed to represent rules and levels of 2D sprite-based games, currently modeling as many as 195 different games. Regular Boardgames (RBG) [3] is another language that combines high-level language features with low-level descriptions, enabling the representation of complex board games. The Ludi system [4], incorporating evolutionary game design, successfully led to the development of the commercially successful game "Yavalath." Moreover, Ludii [5] is a system adopting the "ludemic approach," allowing the decomposition and description of game components at a conceptual level. This enables Ludii to represent over 1,000 traditional games, including board games, card games, dice games, and tile games. Given the broad representational capability and versatility of Ludii GDL, we primarily use Ludii GDL for our research.

Game analysis using Ludii is actively progressing, particularly within board game research [12], [16]–[19]. For instance, methods [16] have been proposed to quantify similarities between board games using concept values defined in Ludii. Moreover, Stephenson *et al.* [17] developed a framework utilizing Ludii to automatically generate rule descriptions for board games. In this study, we focus specifically on generating game descriptions using Ludii GDL.

B. Large Language Models in Games

Since the emergence of ChatGPT [20], large language models (LLMs) have garnered significant attention, prompting exploration into diverse applications within the field of game AI [21], [22]. For example, several methods [23], [24] have been proposed for generating 2D tile-based game levels by fine-tuning GPT-2 [11]. Additionally, research is progressing on prompt-based LLM methods for level generation [25] and quest generation in role-playing games [26]. Dreamcraft [27] further demonstrates a technique to create 3D game objects within Minecraft from textual prompts. Li *et al.* [28] introduces a virtual pet game that achieves real-time gameplay even with smaller LLMs by employing a domain-specific distillation approach.

On the other hand, research has also advanced in automatically generating game rules and descriptions using LLMs and GDL. GAVEL [29] employs an LLM fine-tuned on Ludii game descriptions as a mutation operator in evolutionary search, aiming to create novel games. The objective of GAVEL differs from our research, which focuses on generating game descriptions that maintain consistency with natural language text. Hu *et al.* [9] proposes a method for generating both rules and levels in VGDL using LLMs. LLMaker [30] improves content consistency by utilizing function calling, but its creativity is limited to the scope of the defined functions. GGDG [10] utilizes iterative decoding based on the grammar of Ludii GDL to enhance the grammatical correctness of generated game descriptions. While these previous studies primarily focus on novelty or grammatical correctness of game descriptions, our research differs by emphasizing the improvement of game characteristics of generated game descriptions, bringing them closer to the ground truth through reinforcement learning to enhance LLMs.

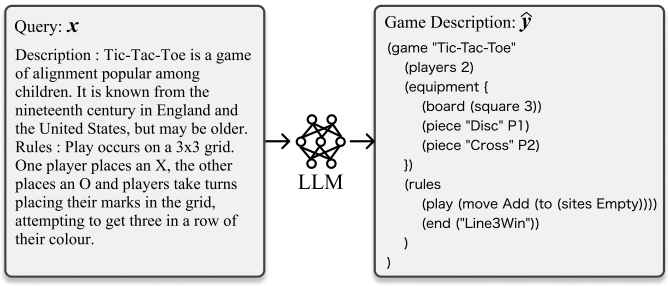

Fig. 2. **An example of GDG for the game Tic-Tac-Toe.** x is text that explains games in natural language. $\hat{y}$ is the game description generated by the LLM in Ludii GDL, a Game Description Language.

C. RL for LLMs

RL-based fine-tuning of LLMs [13]–[15], [31] has been attracting attention. Reinforcement Learning with Human Feedback (RLHF) [31] aligns model behaviors with human preferences by using human feedback as rewards. Recent methods [13], [14] have explored accuracy-based reward functions without human feedback, notably improving logical reasoning tasks such as mathematics and programming. RL-finetuned models are known to exhibit enhanced reasoning capabilities [14]. Additionally, advancements in RL algorithms led to the development of Group Relative Policy Optimization (GRPO) [32], a variant of Proximal Policy Optimization (PPO) [33] optimized for fine-tuning LLMs. In this study, we focus on GDG and define a reward function based on both grammatical correctness and conceptual relevance to the game in generated descriptions, employing GRPO to fine-tune the LLM.

III. Problem Setting

In this section, we define GDG, the primary task addressed in this paper. As shown in Fig. 2, the task involves providing a query x to a large language model (LLM), which then generates the corresponding game description y. The query x is a natural language sentence describing the content or rules of a game. Our goal is to make the game description $\hat{y}$, generated by the LLM, as close as possible to the ground-truth game description y. The acquisition of query x and ground truth y is described in Section V-A.

IV. Methodology

A. Training Procedure

Our training procedure consists of two steps. First, we perform SFT using the paired data of queries and corresponding

game descriptions. Then, we conduct RLFT starting from the SFT-trained model.

SFT in the first step allows LLMs to avoid unstable outputs commonly encountered in the initial stages of RL. For example, to simplify the extraction of generated programs, we instruct the model to output in a specific format, such as `<program>...</program>`. Models without SFT, especially those with smaller parameter sizes such as 1.5B, often ignore the format instructions, generating extraneous text or incorrect formats like ```` ```xml (program)...``` ````. By resolving these formatting issues through the SFT process, the subsequent RL step can focus solely on improving the quality of the game description.

For RLFT, we use GRPO. This algorithm is one of the prominent RL methods for LLMs and has been adopted in DeepSeek-R1 [14], an LLM renowned for its strong reasoning abilities. GRPO extends PPO by using rewards from multiple sampled output candidates $\{o_1, o_2, ..., o_G\}$ for each query, rather than relying on a value function. This approach removes the necessity of value function approximation, thereby enhancing training stability. We discuss reward modeling methods for GRPO in the following subsection.

B. Reward Modeling for GDG

We design two types of rewards: grammar rewards and concept rewards. By employing both, the model can improve not only grammatical accuracy but also the fidelity to the ground truth in the game concept.

a) Grammar rewards: Grammar rewards measure how much of the output $\hat{y}$ is grammatically valid according to the GDL grammar. Using the Earley parser implemented in Ludii [34], we parse $\hat{y}$ from the beginning based on the GDL grammar and obtain the longest grammatically correct substring $\hat{y}_{\text{valid}}$. Let $L_{\hat{y}}$ be the string length of $\hat{y}$ and $L_{\hat{y}_{\text{valid}}}$ be the string length of $\hat{y}_{\text{valid}}$. The grammar reward r_g is calculated as follows:

$$r_g = \frac{L_{\hat{y}_{\text{valid}}}}{L_{\hat{y}}}. \tag{1}$$

When the length of $\hat{y}_{\text{valid}}$ equals that of $\hat{y}$, it reaches its maximum value 1. This reward scale ranges from 0 to 1, approaching 1 as the grammatically valid portion increases.

b) Concept rewards: Concept rewards evaluate the similarity between the game features of the predicted output $\hat{y}$ and those of the ground truth y. An overview of the calculation process for the concept reward is shown in Fig. 3. First, we determine whether the generated game is functional. To clarify this, we introduce two notions: Compatibility indicates whether the Ludii game engine can parse and compile the game without errors, and Functionality indicates whether the compiled game works well enough to be played. If rules fail to function properly, such as when a player cannot move their pieces, the output is considered non-functional. Only compilable games can be evaluated for functionality. A functional game proceeds to the next evaluation step; otherwise, the reward r_c is set to 0. In the next step, we compute the concept values for the output $\hat{y}$ and compare them with the concept values of y. Concept values represent features of a game and were introduced in [12]. Although there are hundreds of concept values, as an initial exploration of incorporating game concepts into RL, we evaluate the following five items, taking inspiration from GAVEL [29].

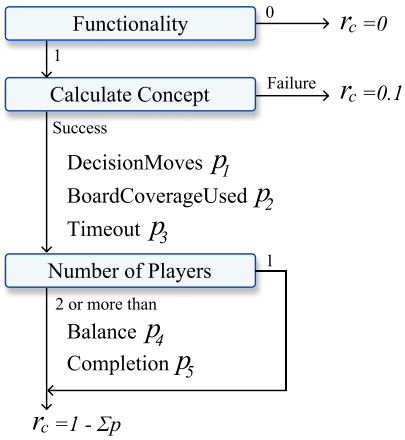

Fig. 3. **Overview of the calculation process for the concept reward.** First, functionality is evaluated, and the concept is calculated only for functional outputs. If the concept can be calculated, penalties p_i are computed across three or five items, depending on the number of players, to derive the final reward r_c.

1) DecisionMoves c_1: Percentage of turns where there was more than one possible move.
2) BoardCoverageUsed c_2: Percentage of used board sites on which a piece was placed at some point.
3) Timeout c_3: Percentage of games that end via timeout.
4) Balance c_4: Similarity between player win rates.
5) Completion c_5: Percentage of games that have a winner (not a draw or timeout).

Each item is cited from the concept definitions of the Ludii concept search [35]. Items 1, 2, and 3 are measured for all functional games, while items 4 and 5 are measured only for games with two or more players. These values are extracted from automatic playouts under a random policy, based on [12], [16]. Following prior studies [10], [29], we perform 50 playouts for the ground truth game and 10 playouts for the predicted game. Let the concept values from the output $\hat{y}$ be denoted as $\hat{c}$. We compute the penalties for each item using a Gaussian kernel:

$$p_i = 1 - \exp\Big(-\frac{1}{2}\big(\tfrac{\hat{c}_i - c_i}{\sigma}\big)^2\Big), \quad 1 \le i \le 5, \tag{2}$$

where $\sigma = 0.3$. By employing a Gaussian kernel, the scale of each penalty is normalized to a range from 0 to 1. The penalty approaches 1 as the features of the output $\hat{y}$ deviate further from those of y. We then use a weighted sum of these penalties to compute the reward:

$$r_c = 1 - \sum_{i=1}^{5} w_i \, p_i. \tag{3}$$

Here, w denotes the weights. When the game is functional but the game features of $\hat{y}$ and y differ greatly—that is, when $p_1, \ldots, p_5$ are all equal to 1.0—we grant a small reward of $r_c = 0.1$ for simply functioning. We implement this by setting every w to 0.18. Furthermore, if the five concept values cannot be computed—for example, when the game is functional but the automatic playout calculation exceeds the timeout limit—we also set r_c to 0.1. Note that the timeout is set to 180 seconds.

Finally, we combine the three rewards as follows:

$$r = r_g + \lambda_c r_c, \tag{4}$$

where λ_c is a scaling parameter, and is set to 1.0.

V. Experiments

A. Datasets

We use only game instances from Ludii-1.14 [36] whose game description y has a token length of 500 or less for both training and evaluation. Token length is calculated using the tokenizer of Qwen2.5-1.5B-Instruct [37]. Our evaluation is performed on 100 randomly selected instances, and the other 410 instances are used for training.

The query x consists of metadata provided by the Ludii game system [36], specifically description and rules. "Description" gives an overview of the game, while "Rules" detail the game's specific rules. Following prior work [10], [29], and to enhance the dataset's generality, we use a game description y in which the game-specific functions defined within each game are fully expanded. After expansion, the game descriptions rely exclusively on primitive functions, omitting Ludii's meta-language features such as definitions, options, rulesets, ranges, and constants.

B. Comparison Methods

Here are the methods we compare in our experiments:

- **GDG**: A baseline approach using LLM's ICL without SFT or RL to generate game descriptions. The prompt includes demonstration examples $(x^{(i)}, y^{(i)})_{i=1}^{N}$, each consisting of a query and corresponding game description. We set $N = 3$.
- **GGDG** [10]: A baseline method designed to improve the grammatical correctness of the generated output $\hat{y}$ based on Ludii GDL grammar. It builds upon GDG by iteratively refining $\hat{y}$ through grammar-based decoding.
- **SFT+GDG**: A baseline approach where the LLM is fine-tuned using SFT alone. The model is trained on the dataset described in Section V-A.
- **RLGDG (ours)**: Our proposed approach, which employs an LLM first fine-tuned with SFT and then further fine-tuned with RL. In RLFT, we use the same dataset as the SFT step, as described in Section V-A.

The GDG and GGDG methods perform few-shot inference using demonstration examples, whereas SFT+GDG and RLGDG perform zero-shot inference without demonstration examples.

System:

A conversation between User and Assistant. The user asks a query, and the assistant correctly reason through a query to generate the appropriate Ludii game program. Write only Ludii game program based on the query in the given task. The answer should be enclosed within <program> </program> tags, i.e., <program> answer here </program>.

User:

Description : Tic-Tac-Toe is a game of alignment popular among children. It is known from the nineteenth century in England and the United States, but may be older.
Rules : Play occurs on a 3x3 grid. One player places an X, the other places an O and players take turns placing their marks in the grid, attempting to get three in a row of their colour.

Assistant:

```
<program>
(game "Tic-Tac-Toe"
   (players 2)
   (equipment {
      (board (square 3))
      (piece "Disc" P1)
      (piece "Cross" P2)
   })
   (rules
      (play (move Add (to (sites Empty))))
      (end ("Line3Win"))
   )
)
</program>
```

Fig. 4. **Our prompts for SFT and RLFT.**

C. Implementation Details

We utilize Qwen2.5-1.5B-Instruct [37] as the LLM. This choice is motivated by two primary reasons: first, we select an open-source LLM to ensure reproducibility of the research results. Second, the chosen model size enables practical fine-tuning with available computational resources. Our experiments are carried out using two NVIDIA RTX 6000 Ada GPUs.

Both SFT and GRPO methods employ full-parameter fine-tuning. Specifically, for the SFT method, we set the sequence length to 768 tokens, batch size to 2, learning rate to 1e-4, and warmup ratio to 0.03, conducting training for a total of 3 epochs.

For GRPO, the prompt length is fixed at 256 tokens, and the completion length at 512 tokens. We use a batch size of 1, a learning rate of 3e-6, and a warmup ratio of 0.1. Furthermore, the number of generated outputs per query G is set to 4. We conduct training for a single epoch, using a temperature of 0.9 for sampling output candidates. Our prompts are shown in Fig 4.

The Earley Parser is implemented using the Lark library [38]. The grammar for parsing Ludii descriptions strictly adheres to the specifications provided in the Ludii Language Reference.

VI. Experimental Results

A. Evaluation Metrics

To evaluate the generated game descriptions, we employ the following metrics based on prior work [10]. For details on Compilability and Functionality, see Section IV-B.

- **Compilability**: The proportion of games that can be successfully parsed and compiled by the Ludii game engine. This score is normalized to a range from 0 to 100.
- **Functionality**: The proportion of games considered playable. This score is normalized to a range from 0 to 100.
- **ROUGE-L** [39]: A metric used to measure linguistic similarity between the generated outputs and ground truth. Commonly used in program synthesis, higher values indicate greater similarity. This metric does not account

TABLE I
COMPARISON WITH BASELINE METHODS. THE BEST RESULTS ARE IN BOLD.

Method	Compilability↑	Functionality↑	ROUGE-L↑	NCD↓
GDG	24.3±3.0	23.3±2.4	53.0±0.6	0.74±0.04
GGDG	11.7±0.7	10.3±0.7	29.5±3.1	0.81±0.02
SFT+GDG	54.3±1.2	52.0±1.5	60.9±0.2	0.51±0.01
RLGDG (ours)	**71.3±0.9**	**70.3±1.3**	**64.0±0.2**	**0.33±0.01**

for grammatical correctness or game-specific features but solely relies on textual similarity. We report the average ROUGE-L F1 score calculated across all test data.

- **Normalized Concept Distance (NCD)**: A metric to measure the similarity of game features between generated outputs and ground truth. Based on [12], [16], games are represented as concept value vectors, and their cosine distance is calculated to determine NCD. The concept value vector includes semantic features and behavior data from random playouts, such as the proportion of board positions used at least once, or the proportion of turns where at least one legal move exists. It also includes the five concept rewards described in Section IV-B. For ground truth games, 50 playouts are executed, while for generated games, 10 playouts are performed. For non-functional games where the concept distance cannot be calculated, NCD is set to 1. The average NCD is computed to evaluate the quality of GDG.

Experiments are conducted using three different random seeds following [10], and the mean and standard error for each metric are reported.

B. Comparison with Baseline Methods

Table I shows the comparative results with baseline methods. RLGDG outperforms baseline methods across all evaluation metrics. Notable improvements are observed in Compilability and Functionality, demonstrating that RLGDG effectively ensures grammatical correctness and practical playability of the generated game descriptions. Additionally, a significant improvement in NCD suggests that RLGDG enables LLMs to better learn game concepts.

GGDG underperforms compared to GDG across all metrics. This is likely because GGDG's iterative improvement decoding requires handling numerous detailed instructions, a task challenging for small-scale models with approximately 1.5B parameters. In contrast, RLGDG significantly enhances performance using the 1.5B-parameter model, suggesting that it also holds advantages in terms of inference cost efficiency.

C. Ablation Study

Comparison of Reward Modeling. We conduct an ablation study on the reward modeling of RLGDG, and the results are presented in Tab. II. The results demonstrate that using both types of rewards simultaneously yields the best performance across all evaluation metrics.

Focusing on compilability and functionality, we observe that grammar reward accounts for a significant proportion of the performance improvement from SFT+GDG to RLGDG, specifically 86.5% for compilability and 87.7% for functionality. This suggests that the grammar reward contributes to enhancing grammatical accuracy.

Introducing the concept reward results in an additional improvement of 10.8% in NCD compared to RLGDG without the concept reward. This indicates that the inclusion of the concept reward not only helps generate compilable and functional code but also improves the generation of game descriptions that more precisely capture the ground truth game concepts.

Game Category of Test Games. We investigate the performance comparison across different game categories for test instances. Following the methodology of previous research (GGDG), we compared five categories: racing games (board/race), mancala games (board/sow), puzzle games (puzzle), line games (board/space/line), and war games, including capture games (board/war). The test instances used here are extracted from the instances used in Section V-A. The results are summarized in Tab. III. RLGDG outperforms SFT+GDG in all metrics across all categories.

For comparing categories, SFT+GDG demonstrates the lowest performance in the puzzle category and the highest performance in the board/space/line category across all metrics. We believe this performance difference arises from the varying amounts of training data. Table IV summarizes the number of training instances in each category, and the average concept distance from the board/space/line category, which has the most training instances. As the board/space/line category has the largest number of instances, it is considered easier for the model to learn from. In contrast, the puzzle category has the fewest instances, equal in number to the board/race category. When compared to board/race games, puzzle games are conceptually farther from board/space/line than board/race games. Therefore, it is easier for the model to transfer insights gained from the board/space/line category to the board/race category than to the puzzle category, leading to lower performance in the puzzle category.

For comparing methods, RLGDG improves compilability by 66.5%, functionality by 100%, and other metrics compared to SFT+GDG in the puzzle category. Furthermore, in the board/space/line category, where SFT+GDG already demonstrated high performance, RLGDG further improved performance, achieving an NCD of 0.14. These results suggest that RLGDG may overcome the limitations of SFT independently of the category or the amount of training data.

VII. QUALITATIVE ANALYSIS

Comparison with Baseline Methods. We conduct a qualitative analysis comparing the best-performing baseline method, SFT+GDG, against our proposed method, RLGDG. Figure 5 shows the generation results for Tic-Tac-Mo, a game that extends the player count of Tic-Tac-Toe to three players. The result from SFT+GDG is non-compilable and non-functional due to the `trigger ''End'' Mover Win`. According to the grammar rules of Ludii GDL, `trigger ''End'' Mover` should only be followed by a `then` clause. Additionally, the

TABLE II
COMPARISON OF REWARD MODELING IN OUR PROPOSED METHOD. THE BEST RESULTS ARE IN **BOLD**.

Method	Grammar	Concept	Compilability↑	Functionality↑	ROUGE-L↑	Normalized Concept Distance↓
SFT+GDG [10]			54.3±1.2	52.0±1.5	60.9±0.2	0.51±0.01
RLGDG w/o concept	✓		69.0±1.7	66.3±1.8	**64.0±0.1**	0.37±0.02
RLGDG	✓	✓	**71.3±0.9**	**70.3±1.3**	**64.0±0.2**	**0.33±0.01**

TABLE III
COMPARISON OF TEST INSTANCE CATEGORIES.

Method	Compilability↑	Functionality↑	ROUGE-L↑	NCD↓
board/race				
SFT+GDG	63.0±9.8	59.3±9.8	51.8±2.0	0.45±0.09
RLGDG	81.5±3.7	74.1±7.4	53.4±0.7	0.29±0.06
board/sow				
SFT+GDG	36.1±7.3	36.1±7.3	71.1±1.2	0.66±0.07
RLGDG	58.3±16.7	58.3±16.7	71.7±1.6	0.45±0.15
puzzle				
SFT+GDG	16.7±4.8	11.1±7.3	40.6±1.5	0.90±0.07
RLGDG	27.8±10.0	22.2±11.1	43.0±2.3	0.80±0.10
board/space/line				
SFT+GDG	76.2±3.1	75.0±2.1	73.9±0.5	0.27±0.02
RLGDG	89.3±5.5	88.1±6.6	76.7±0.6	0.14±0.06
board/war				
SFT+GDG	66.7±0.0	66.7±0.0	58.8±0.3	0.38±0.00
RLGDG	88.9±0.0	83.3±3.2	64.8±0.6	0.23±0.03

TABLE IV
NUMBER OF TRAINING INSTANCES IN EACH CATEGORY AND THE AVERAGE CONCEPT DISTANCE FROM THE LARGEST CATEGORY.

Category	Number of Instances	Distance from board/space/line
board/race	13	0.068
board/sow	33	0.081
puzzle	13	0.098
board/space/line	113	0.000
board/war	65	0.059

description regarding the termination conditions, marked with a red rectangle, should be placed within the `end` clause of the rules. In contrast, the result obtained with RLGDG is compilable and functional, and the output game description closely matches the ground truth. The only deviation from the ground truth is the board specification, which is `rectangle 1 5`, identical to the one from SFT+GDG. Using different seeds, it can sometimes result in `rectangle 3 5`, in which case it perfectly matches the ground truth.

Figure 6 shows the generated results for Yavalath. Yavalath is a game developed by the Ludi system [4], where the objective is to align four markers of the same player in a straight line without first aligning three markers. The results from SFT+GDG are compilable and functional, with an NCD of 0.012. However, there are primarily three incorrect predictions: (i) Introduction of phases not described in Yavalath's rules, (ii) inclusion of a move called `moveAgain`, which allows placing a second marker within the same turn during the `Play` phase, although no such rule exists in Yavalath, and (iii) omission of the termination condition where aligning exactly three markers in a straight line results in a loss. In contrast, the results from RLGDG are compilable and functional, achieving an NCD of 0.006, indicating that the resulting game concept is closer to the ground truth compared to the SFT+GDG results. While the ground truth includes rules such as `(rotate 90 ...)` and `(meta (swap))`, these instructions were not included in the input text. Additionally, the number of players is different, but both two and three-player options are mentioned in the input text. Thus, the game rules described in the text are sufficiently covered by the RLGDG output.

Analysis of Failure Cases. As an example of a failure case, we analyze the generation results for the Knight's Tour puzzle from the puzzle category, as illustrated in Fig. 7. In this game, the knight is moved across a chessboard so that it visits every square exactly once.

SFT+GDG generates an output that could not be compiled. This was because after the expression `(if (isEven (count Moves))`, a function returning a boolean value was expected. However, it was actually followed by the in-game action `(move Add (to (sites Empty)) copy:True)`.

The output from RLGDG is compilable and functional but contains errors in the following three aspects: (i) In the ground truth, the knight's initial position can be any available square during the placement phase. However, the output from RLGDG restricts the knight's initial placement to the central square of the board. (ii) The RLGDG output lacks the termination condition at the end of the game, where a win occurs if the knight successfully visits all squares exactly once. (iii) The knight's movement pattern generated by RLGDG is incorrect. `F` indicates moving forward by one step, and `R` means rotating 90 degrees to the right. In the RLGDG output, the sequence `FFR` turns the knight right after moving forward two steps. However, the correct movement for the knight would require an additional forward step, making the correct sequence `FFRF`. This error likely arises from the limited availability of training data related to the puzzle category, as discussed in Section VI-C.

VIII. DISCUSSION AND CONCLUSION

In this study, we propose Reinforcement Learning-based fine-tuning of LLMs for Game Description Generation (RLGDG). Existing approaches have primarily focused on

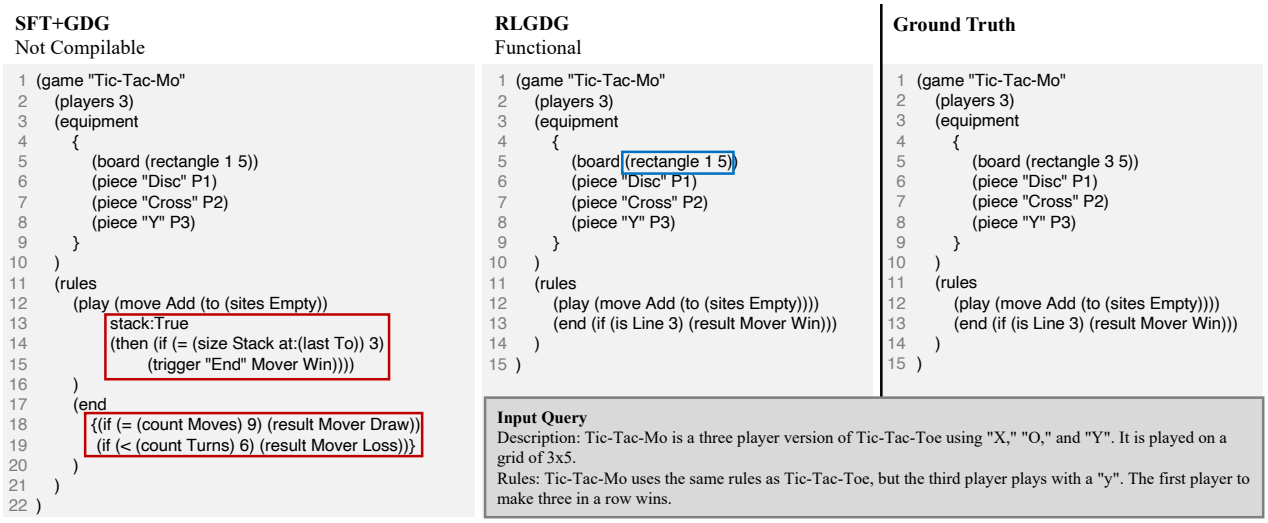

Fig. 5. **Comparison of generation results with baseline methods for Tic-Tac-Mo.** The part enclosed in red differs from the ground truth.

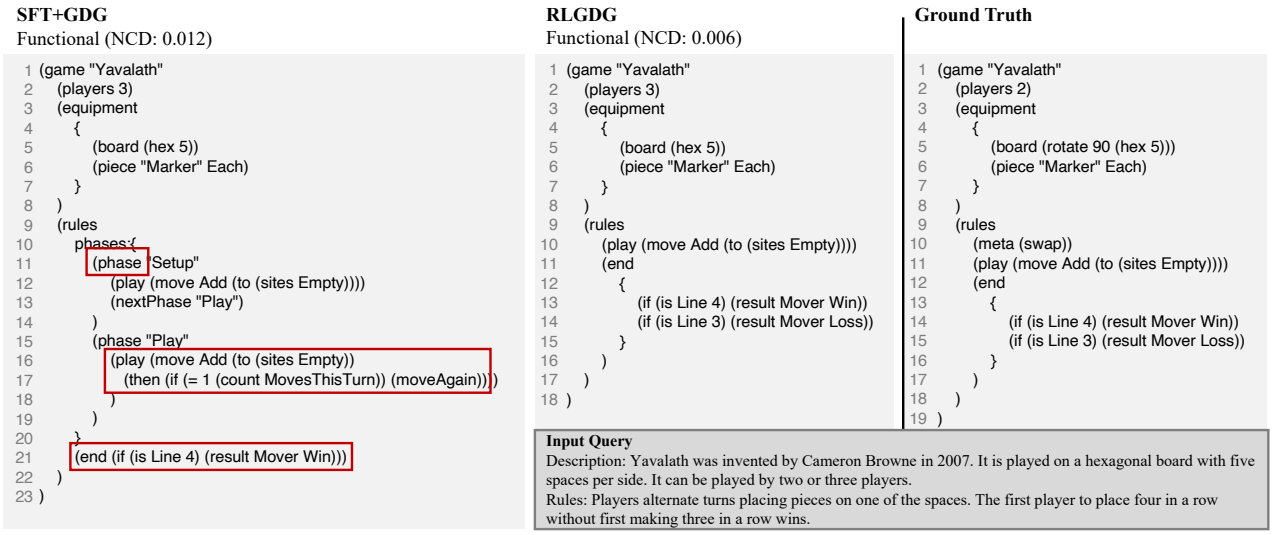

Fig. 6. **Comparison of generation results with baseline methods for Yavalath.** The incorrectly predicted parts are enclosed in red.

improving grammatical accuracy; however, our method simultaneously enhances both grammatical correctness and conceptual fidelity to game concepts. Specifically, we introduce grammar and concept rewards and adopt a two-stage training strategy that applies RL after Supervised Fine-Tuning (SFT). Experimental results demonstrated that our proposed method achieved superior performance compared to baseline methods in terms of both grammatical accuracy and conceptual fidelity. However, improvements are limited in categories with insufficient training data. Future research directions for training data include synthetic data generation [40], data augmentation using evolutionary algorithms [29], and leveraging larger-scale language models. Moreover, extending the concept reward from five to the full set of concept values remains a key target. These studies are expected to generate high-quality game descriptions from natural language, thereby supporting designers and engineers in AI-driven game development.

ACKNOWLEDGMENTS

This work was supported by JST, ACT-X Grant Number JPMJAX23CE, Japan.

REFERENCES

[1] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," 2008.
[2] T. Schaul, "A video game description language for model-based or interactive learning," in *Proc. of CIG*, 2013.
[3] J. Kowalski, M. Mika, J. Sutowicz, and M. Szykuła, "Regular boardgames," in *Proc. of AAAI*, 2019.
[4] C. Browne, *Evolutionary Game Design*. Springer, 2011.
[5] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne, "Ludii – the ludemic general game system," in *Proc. of ECAI*, 2020.
[6] T. S. Nielsen, G. A. B. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with vgdl," in *Proc. of CIG*, 2015.

SFT+GDG
Not Compilable

```
(game "Knight's Tour"
  (players 1)
  (equipment
    {
      (board (square 8))
      (piece "Knight"
        P1
        (move
          Leap
          {{F F R F}{F L F F}}
          forward:True
          (to if:(is Empty (to)))
          (then (if (is Enemy (who at:(to))) (remove (to))))
        )
      )
    }
  )
  (rules
    (start { (place "Knight" (sites Centre)) })
    (play
      (if
        (is Even (count Moves))
        (move Add (to (sites Empty)) copy:True)
        (forEach Piece)
      )
    )
    (end
      (if
        (and (= (what at:(last To)) P1) (no Pieces Next))
        (result Mover Win)
      )
    )
  )
)
```

RLGDG
Functional

```
(game "Knight's Tour"
  (players 1)
  (equipment
    {
      (board (square 8))
      (piece "Knight" P1
        (move
          Leap
          {{F F R}{F L F}}
          forward:True
          (to if:(is Empty (to)))
        )
      )
    }
  )
  (rules
    (start { (place "Knight" (sites Centre)) })
    (play (forEach Piece))
    (end (if (no Moves Next) (result Mover Win)))
  )
)
```

Input Query
Description: Knight's tour is a puzzle by which a Chess knight is moved on a board so that is is placed in every square on the board only once. It has been documented in India, where the movement of the horse piece in Chaturanga has the same movement as the Chess knight.
Rules: Played with one knight on a Chess board. The goal is to move the knight onto every square of the board only once using its typical move as in Chess.

Ground Truth

```
(game "Knight's Tour"
  (players 1)
  (equipment
    {
      (board (square 8))
      (piece "Knight" P1)
    }
  )
  (rules
    phases:{
      (phase "Placement"
        (play (move Add (to (sites Empty))))
        (nextPhase "Movement")
      )
      (phase "Movement"
        (play
          (move
            Leap
            (from (last To))
            {{F F R F}{F F L F}}
            (to if:(is Empty (to)))
            (then (add (to (last From))))
          )
        )
      )
    }
    (end
      {
        (if (>= (count Moves) (count Sites "Board")) (result P1 Win))
        (if (no Moves P1) (result P1 Loss))
      }
    )
  )
)
```

Fig. 7. **Comparison of generation results for Knight's Tour.** Parts of the ground truth that are missing from or incorrectly predicted in the RLGDG output are indicated by boxes.

[7] A. Khalifa, M. C. Green, D. Perez-Liebana, and J. Togelius, "General video game rule generation," in *Proc. of CIG*, 2017.

[8] T. Maurer and M. Guzdial, "Adversarial random forest classifier for automated game design," in *Proc. of FDG*, 2021.

[9] C. Hu, Y. Zhao, and J. Liu, "Game generation via large language models," in *Proc. of CoG*, 2024.

[10] T. Tanaka and E. Simo-Serra, "Grammar-based Game Description Generation using Large Language Models," *IEEE Trans. Games.*, 2024.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," 2019.

[12] E. Piette, M. Stephenson, D. J. Soemers, and C. Browne, "General board game concepts," in *Proc. of CoG*, 2021.

[13] L. Trung, X. Zhang, Z. Jie, P. Sun, X. Jin, and H. Li, "ReFT: Reasoning with reinforced fine-tuning," in *Proc. of ACL*, 2024.

[14] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," 2025.

[15] A. Kumar, V. Zhuang, R. Agarwal, Y. Su, J. D. Co-Reyes, A. Singh, K. Baumli, S. Iqbal, C. Bishop, R. Roelofs *et al.*, "Training language models to self-correct via reinforcement learning," 2024.

[16] M. Stephenson, D. J. N. J. Soemers, E. Piette, and C. Browne, "Measuring board game distance," in *Proc. of Computer and Games*, 2022.

[17] M. Stephenson, E. Piette, D. J. N. J. Soemers, and C. Browne, "Automatic generation of board game manuals," in *Proc.of Advances in Computer Games*, 2021.

[18] M. Stephenson, D. J. N. J. Soemers, E. Piette, and C. Browne, "General game heuristic prediction based on ludeme descriptions," in *Proc. of CoG*, 2021.

[19] D. J. N. J. Soemers, Éric Piette, M. Stephenson, and C. Browne, "The ludii game description language is universal," 2024.

[20] OpenAI, Nov 2022. [Online]. Available: https://openai.com/blog/chatgpt

[21] R. Gallotta, G. Todd, M. Zammit, S. Earle, A. Liapis, J. Togelius, and G. N. Yannakakis, "Large language models and games: A survey and roadmap," in *arXiv preprint arXiv:2402.18659*, 2024.

[22] M. U. Nasir and J. Togelius, "Practical pcg through large language models," in *Proc. of CoG*, 2023.

[23] G. Todd, S. Earle, M. U. Nasir, M. C. Green, and J. Togelius, "Level generation through large language models," in *Proc. of FDG*, 2023.

[24] S. Sudhakaran, M. González-Duque, M. Freiberger, C. Glanois, E. Najarro, and S. Risi, "MarioGPT: Open-ended text2level generation through large language models," in *Proc. of NeurIPS*, 2023.

[25] F. Abdullah, P. Taveekitworachai, M. F. Dewantoro, R. Thawonmas, J. Togelius, and J. Renz, "The 1st ChatGPT4PCG competition," 2024, pp. 1–17.

[26] S. Värtinen, P. Hämäläinen, and C. Guckelsberger, "Generating role-playing game quests with gpt language models," *IEEE Trans. Games.*, vol. 16, no. 1, pp. 127–139, 2024.

[27] S. Earle, F. Kokkinos, Y. Nie, J. Togelius, and R. Raileanu, "Dreamcraft: Text-guided generation of functional 3d environments in minecraft," in *Proc. of FDG*, 2024.

[28] J. Li, Y. Li, N. Wadhwa, Y. Pritch, D. E. Jacobs, M. Rubinstein, M. Bansal, and N. Ruiz, "Unbounded: A generative infinite game of character life simulation," in *Proc. of ICLR*, 2025.

[29] G. Todd, A. Padula, M. Stephenson, É. Piette, D. J. Soemers, and J. Togelius, "Gavel: Generating games via evolution and language models," in *Proc. of NeurIPS*, 2024.

[30] R. Gallotta, A. Liapis, and G. Yannakakis, "Consistent game content creation via function calling for large language models," in *Proc. of CoG*, 2024.

[31] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," in *Proc. of NeurIPS*, 2022.

[32] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," 2024.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[34] J. Earley, "An efficient context-free parsing algorithm," *Commun. ACM*, vol. 13, no. 2, p. 94–102, 1970.

[35] Digital Ludeme Project, "Ludii concept search." [Online]. Available: https://ludii.games/searchConcepts.php

[36] ——, "Ludii portal." [Online]. Available: https://ludii.games/index.php

[37] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei *et al.*, "Qwen2. 5 technical report," 2024.

[38] Digital Ludeme Project, "Lark parser." [Online]. Available: https://github.com/lark-parser/lark

[39] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Proc. of Text Summarization Branches Out*, 2004.

[40] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-instruct: Aligning language models with self-generated instructions," in *Proc. of ACL*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Jul. 2023.