Grammar-based Game Description Generation using Large Language Models

Tsunehiko Tanaka, Edgar Simo-Serra

***Abstract*—Game Description Language (GDL) provides a standardized way to express diverse games in a machine-readable format, enabling automated game simulation, and evaluation. While previous research has explored game description generation using search-based methods, generating GDL descriptions from natural language remains a challenging task. This paper presents a novel framework that leverages Large Language Models (LLMs) to generate grammatically accurate game descriptions from natural language. Our approach consists of two stages: first, we gradually generate a minimal grammar based on GDL specifications; second, we iteratively improve the game description through grammar-guided generation. Our framework employs a specialized parser that identifies valid subsequences and candidate symbols from LLM responses, enabling gradual refinement of the output to ensure grammatical correctness. Experimental results demonstrate that our iterative improvement approach significantly outperforms baseline methods that directly use LLM outputs. Our code is available at https://github.com/tsunehiko/ggdg**

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

I. Introduction

A Game Description Language (GDL) [1]–[5] is a domain-specific language that expresses a wide range of games in a unified notation. For example, Ludii GDL [5] models over 1,000 games, primarily board games, as shown in Fig. 1. Game descriptions represented in GDLs are highly machine-readable, making it easy to simulate gameplay using dedicated game engines. Given the amenability of GDLs for automatic game evaluation, they have been extensively used in research on automated game design. In particular, search-based methods such as evolutionary algorithms [4], MCTS [6], [7], and random forests [8] have proven successful in generating game descriptions. Most research primarily focused on mutating existing games based on fitness functions to generate novel games. However, the task of generating game descriptions from natural language texts has not yet been sufficiently explored, and has the potential to lower the bar of entry to game design to non-specialists. In this research, we use Large Language Models (LLMs) [9], [10], which excel at understanding textual context, to generate game descriptions from natural language text in a two-stage process to enforce grammatical correctness.

LLMs are language models with an enormous number of parameters, pre-trained on vast amounts of text data. These models possess the ability to solve various tasks without additional training [11], [12], and this ability can be elicited by including the task context in the prompt, a technique known as In-Context Learning (ICL) [13]. Hu *et al*. [14] applied this capability to game description generation by incorporating explanations for GDL notations and examples of game descriptions in the prompt context. Their results have shown that more accurate game descriptions can be generated by appropriately refining the prompt context. However, LLMs may still generate grammatically incorrect game descriptions. Such grammatically inaccurate game descriptions cannot be correctly parsed and loaded by game engines, making it difficult to evaluate them through gameplay simulation. To generate higher-quality games, it is important first to ensure that LLMs can produce grammatically correct GDL game descriptions.

This paper presents a method for LLMs to generate more grammatically accurate game descriptions. We propose an approach to iteratively improve LLMs' initial responses using the GDL grammar. Our generation framework consists of two stages: (i) generating the minimal grammar required for game descriptions, and (ii) iteratively improving the game description based on this minimal grammar. First, we use LLMs to generate the minimal grammar required to produce the game description, making use of the GDL grammar. Next, a parser based on the minimal grammar determines grammatically valid subsequences and a set of candidate symbols that could follow these subsequences from the LLMs' responses. LLMs

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

x :
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.

y :
```
(game "Tic-Tac-Toe"
   (players 2)
   (equipment
      {
         (board (square 3))
         (piece "Disc" P1)
         (piece "Cross" P2)
      }
   )
   (rules
      (play (move Add (to (sites Empty))))
      (end (if (is Line 3) (result Mover Win)))
   )
)
```

Fig. 1. **An example of Ludii game description for the game "Tic-Tac-Toe".** x is text that explains games in natural language. y is a game description in Ludii GDL, a Game Description Language.

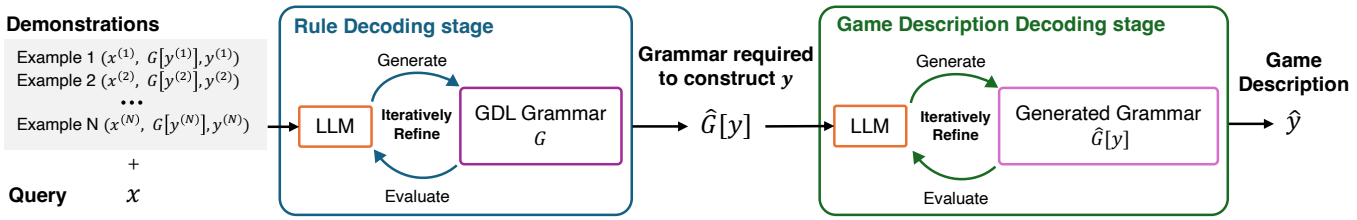

Fig. 2. **An overview of our framework for grammar-based game description generation.** We generate the game description y from a natural language query x using Large Language Models (LLMs). First, we generate the grammar $\hat{G}[y]$ required to construct y in the Rule Decoding stage, and then generate $\hat{y}$ based on $\hat{G}[y]$ in the Game Description Decoding stage. The core of our framework is iteratively decoding by leveraging the grammar of game description languages (GDLs) to improve the initial response from LLMs. We make use of Ludii GDL as our GDL, which can model a larger variety of games and is a context-free grammar.

then re-infer the missing parts based on these subsequences and candidate symbols, gradually generating grammatically accurate game descriptions. Experimental results demonstrate that our framework is more effective in generating game descriptions compared to a baseline that directly uses the LLMs' initial responses as an output.

Our contributions can be summarized as follows:

- We propose a framework for generating game descriptions from natural language text by using LLMs and GDLs.
- Our framework incorporates GDL grammar into the generation process and iteratively improves the grammatical correctness of the LLM's output.
- We propose iterative improvement decoding methods specialized for grammar generation and game description generation, respectively.
- We demonstrate the effectiveness of our framework through extensive experiments on game description generation.

II. Related Work

A. Game Description Language

A Game Description Language (GDL) is a domain-specific language for games. GGP-GDL [1] was introduced in 2005 and has become the standard in General Game Playing, where artificial agents are developed to play a wide variety of games. After GGP-GDL, various types of GDLs have been developed. VGDL [2] is a language that represents both the levels and rules of 2D sprite-based games, and it models 195 games. RBG [3] models complex board games by combining low-level and high-level languages. Ludii [5] is a game system developed based on the ludemic approach, which decomposes games into conceptual units of game-related information. Ludii models more than 1,000 traditional games, including board games, card games, dice games, and tile games. Ludii's grammar is a Context-Free Grammar (CFG) in Extended Backus-Naur Form (EBNF) style [15]. Due to its ability to generalize and model more games, as well as its capacity to express complete game descriptions in CFG, we use Ludii GDL as our GDL. Note that our approach can be applied to other GDLs that follow CFG, *e.g.*, the rule section of VGDL.

Diverse analyses of games using Ludii, especially focusing on board games, have been conducted [16]–[20]. For example, the distance between board games using concept values extracted from Ludii has been proposed in [16]. Stephenson *et al.* [17] have presented a framework for automatically generating board game manuals using Ludii. Unlike these works, we add a new perspective to the analysis by focusing on game description generation using grammar.

B. Automated Game Design

Automated game design [21], [22] is one of the core themes in the field of game AI. As AI technology rapidly advances, various aspects of how AI can be utilized in automated game design have been discussed, including the design process [23], design patterns [24], and creative machine learning [25]. Deep learning-based ML is often used in the domain of procedural content generation for level design [26]–[31]. Several studies [32], [33] propose methods to generate a game by integrating multiple elements such as visuals, audio, narrative, levels, rules, and gameplay. Generating games using new representations such as answer set programming [34] and game graphs [35] has also been explored. Word2World [36] uses LLMs to design from stories to playable games procedurally. These approaches design games without including GDLs.

Automated game design for generating game designs in the GDL format has been explored [4], [6]–[8], [37]. The Ludi system [4], which was the precursor to the ludemic approach used in later Ludii [5], employed evolutionary game design and generated the commercially viable game Yavalath. Thorbjørn *et al.* [6] explored the approach to generate VGDL [2] for arcade games using evolutionary algorithms. GVG-RG [7] proposes a framework for generating appropriate game rules for given game levels. Cicero [37] is a mixed-initiative tool that assists in prototyping 2D sprite-based games using VGDL. Thomas *et al.* [8] aimed to acquire a fitness function that guides game design generation using adversarial random forest classifiers. These approaches generate novel games based on existing games, but our approach differs in that it generates game descriptions from natural language text.

C. Large Language Models in Games

Since the advent of ChatGPT [38] in late 2022, Large Language Models (LLMs) have attracted significant attention,

and various ways of using LLMs in automated game design have been explored [39], [40]. Several studies [26], [27] have fine-tuned GPT-2 [41] models to generate 2D tile-based game levels. Prompt-based approaches for LLMs have also been proposed for level generation [28]. Other researchers have trained GPT models to generate quests for role-playing games [42]. Dreamcraft [43] is a method that uses LLMs to generate 3D game objects for Minecraft from text prompts. In addition to generation, researchers have focused on evaluating LLMs. Studies have assessed how well prompt-based level generation methods replicate and generalize [44], and have evaluated the capabilities of LLMs in mixed-initiative game design [45].

GAVEL [46] uses LLMs fine-tuned on Ludii's game descriptions as mutation operators in evolutionary search. While GAVEL aims to generate games with high novelty, our goal is to generate game descriptions that align with natural language text. LLMGG [14] generates both the rules and levels of games represented in VGDL using LLMs. In LLMGG, LLMs take a part of the VGDL [2] representation or examples of other games as prompts, which generate a complete VGDL-based game in one step. The authors discuss that incorrect game levels and rules not included in the VGDL grammar are generated. In contrast, our approach involves multi-step generation, and the necessary grammar rules to build a game description are generated as intermediate representations in the middle steps. Moreover, since the game description is iteratively generated based on these grammatical rules, this approach prevents inaccurate syntax and improves consistency.

D. Program Synthesis

In program synthesis, the task of generating programs from natural language is called semantic parsing. Semantic parsing has benefited from advancements in LLMs. Several efforts [11], [12] have already explored generating code in general-purpose programming languages such as Python using LLMs. To improve the accuracy of the generated programs, constrained decoding [47]–[49] has also been studied. Grammar-constrained decoding [50]–[52] restricts the output space of LLMs to a space described by a grammar, enabling the generation of structured outputs like programs. Our framework includes decoding for grammar generation, in addition to grammar-constrained decoding for game descriptions.

Evolutionary algorithms that use LLMs as evolutionary operators [53]–[56] are also gaining attention in program synthesis. Quality-diversity algorithms utilizing LLMs have been proposed for controllable program synthesis, such as neural architecture search [57]. These studies open up the possibility that using LLMs can also improve GDL generation.

In domain-specific language (DSL) generation, a prompting method [58] has been proposed that introduces grammar as an intermediate product while LLMs iteratively reason to solve tasks, also known as a chain-of-thought reasoning [59]. However, in [58], the evaluation targets simple and short DSLs such as SMCalFlow [60] and GeoQuery [61], and the use of LLMs in the approach is limited. We aim to make more effective use of LLMs in our approach and evaluate it with complex and lengthy DSLs that represent game design, such as Ludii [5].

$G[y]$:
```
game ::= "(game" string players equipment rules_rules ")"
string ::= "Tic-Tac-Toe" | "Disc" | "Cross"
players ::= "(players" int ")"
equipment ::= "(equipment" "{" item item item "}" ")"
item ::= container | component
container ::= container_board_board
container_board_board ::= "(board" graph ")"
graph ::= basis
basis ::= square
square ::= "(square" dim ")"
dim ::= int
component ::= component_piece
component_piece ::= "(piece" string role_type ")"
role_type ::= P1 | P2 | MOVER
rules_rules ::= "(rules" play end ")"
play ::= "(play" moves ")"
moves ::= decision
decision ::= move
move ::= "(move" move_site_type moves_to ")"
move_site_type ::= ADD
moves_to ::= "(to" sites ")"
sites ::= "(sites" sites_index_type ")"
sites_index_type ::= EMPTY
end ::= "(end" end_rule ")"
end_rule ::= end_if
end_if ::= "(if" boolean result ")"
boolean ::= booleans_is_is
booleans_is_is ::= "(is" LINE int ")"
result ::= "(result" role_type result_type ")"
result_type ::= WIN
```

Fig. 3. **Minimal Backus-Naur Form (BNF) grammar $G[y]$ for Tic-Tac-Toe.** Redundant rules are omitted for simplicity.

III. Problem Setting

In this section, we define our problem setting. We first describe the generation of games written in GDLs. We next review in-context learning in the game description generation. Notations used in this paper are summarized in Tab. I.

A. Game Description Generation

Let G be the GDL grammar, and let $L(G)$ represent the set of game descriptions generated by G. We call the task of inputting a natural language query x that describes the content and rules of a game, and generating a corresponding game description $y \in L(G)$, *game description generation*. Our goal is to make the generated game description $\hat{y}$ as close as possible to the ground truth y.

In this paper, we use Ludii GDL as our GDL as it is one of the main references in GDL research. Additionally, Ludii's grammar is a CFG in EBNF style, which allows us to generate complete game descriptions based on a CFG. An example of Ludii game description generation is shown in Fig. 1. From this point forward, unless otherwise specified, G represents Ludii's grammar.

B. In-Context Learning

In-Context Learning (ICL) [13] is an efficient method that provides pretrained LLMs with a few task-specific examples to obtain more precise and accurate results. This approach does not require additional training or fine-tuning; instead, it relies on the LLM's ability to identify and apply patterns

TABLE I
GLOSSARY OF NOTATIONS USED IN THIS PAPER.

Notation	Description
G	GDL grammar (Ludii grammar in this paper)
$L(G)$	Set of game descriptions generated by G
x	Texts explaining the rules of a game
y	A game description (ground truth)
$G[y]$	Minimal grammar extracted from G containing only the grammar rules necessary to generate game description y
$(x^{(i)}, y^{(i)})_{I=1}^{N}$	Demonstrations of game description generation to be included in LLMs' prompts
$\hat{G}[y]_{\text{valid}}$	Set of grammar rules in the predicted $\hat{G}[y]$ that are included in G
N_U	Set of non-terminal symbols defined in $G[y]$ but not defined in $\hat{G}[y]_{\text{valid}}$
G_{N_U}	Set of grammar rules in G defining N_U
$G[y]_{N_U}$	Minimal set of rules extracted from G_{N_U} necessary to generate y
$\hat{y}_{\text{valid}}$	Part of $\hat{y}$ that conforms to $\hat{G}[y]$
$\Omega_{\text{candidate}}$	Set of candidate terminal symbols to follow $\hat{y}_{\text{valid}}$
ω	Terminal symbol selected from $\Omega_{\text{candidate}}$ by generators such as LLMs
$\hat{\cdot}$	All hat notations indicate symbols are predictions
$\cdot'$	All prime notations indicate that the symbols are in the LLM's output space. Among these, the one that maximizes the generation probability receives the hat notation.

from the provided examples. In ICL, LLM is conditioned on N demonstrations $(x^{(i)}, y^{(i)})_{i=1}^{N}$ followed by a test example query x, and generates y as $P_{\text{LLM}}(y|(x^{(i)}, y^{(i)})_{i=1}^{N}, x)$. Recent studies [62], [63] have reported that the few-shot performance on complex reasoning tasks can be improved by inserting intermediate reasoning steps between $x^{(i)}$ and $y^{(i)}$ in the demonstrations.

The effectiveness of ICL depends on how effectively the solution to a task can be conveyed through demonstrations. Intuitively, providing more demonstrations to the LLM seems to be beneficial. However, the context length, which is the maximum length that the LLM can capture, is determined during pre-training. Therefore, it is not possible to input a number of demonstrations that exceed this context length. For example, when applying Llama3 [9], one of the leading open-source LLMs, to Ludii's game descriptions, the context length can easily be exceeded. In particular, the average token length of Ludii's game descriptions using the Llama3-8B-Instruct tokenizer is 2,458. The context length of Llama3-8B-Instruct is 8,192, which limits the number of Ludii demonstrations that can be input to a few at most. This limitation when using complex examples such as those described in Ludii is common to many LLMs. Game descriptions generated based on such limited context lack grammatical accuracy and cannot make the games functional and will be discussed in Sec. VI.

IV. METHODOLOGY

In this section, we explain our approach to iteratively improve the initial responses of the LLM. To evaluate the game description and improve its grammatical accuracy, we introduce the minimum grammar $G[y] \subseteq G$ required to construct y. This minimum grammar $G[y]$ is expected to provide more context to LLMs when included in the prompt, thereby enhancing In-Context Learning (ICL). As shown in Fig. 2, our generation process consists of two stages: First, we input the query x of the test example into an LLM to generate the minimum grammar $\hat{G}[y]$ required to construct y. $\hat{G}[y]$ is then evaluated, and the parts that conform to the Ludii grammar G are extracted. The LLM then generates the missing rules. Next, based on $\hat{G}[y]$, the LLM generates the game description $\hat{y}$. $\hat{y}$ is evaluated, and the parts that conform to $\hat{G}[y]$ are extracted. The LLM then infers the rest. Evaluation and LLM generation are repeated at each stage. In this section, we first introduce the minimum grammar $G[y]$, then explain the two-stage generation, and finally describe the decoding process for each stage in detail.

A. Grammar-based Game Description Generation

$G[y]$ is the minimal grammar that extracts only the rules necessary to generate y from all the rules of G. An example of $G[y]$ for tic-tac-toe is shown in Figure 3. $G[y]$ is a subset of the full grammar G, where $y \in L(G[y])$ and $\forall r \in G[y], y \notin L(G[y] \backslash \{r\})$. For any rule r in $G[y]$, removing r makes it impossible to generate y ($y \notin L(G[y] \setminus r)$). $G[y]$ is minimal in the sense that it contains exactly the rules required to generate y, with no superfluous rules. A parser based on this minimal grammar can determine grammatically valid subsequences and the set of candidate symbols that follow them from the LLM's responses. The LLM then generates the rest of the game description based on the subsequences and candidate symbol groups. Intuitively, it is expected that repeating this parsing and generation process will result in a more grammatically accurate game description than the initial response from the LLM. This iterative improvement method is explained in Sec. IV-B.

From the perspective of ICL, providing more context to LLMs is expected to have a positive impact. It is known that providing GDL grammar to LLMs can improve the quality of generated game descriptions, especially in terms of grammatical accuracy [14]. However, the token length of Ludii's grammar using the Llama3-8B-Instruct tokenizer is 15,442, which is longer than the context length of most LLMs. The straight-forward approach of including all the GDL grammar in the prompt as proposed in LLMGG is not feasible without significant more computation power. The average token length of $G[y]$ using the Llama3-8B-Instruct tokenizer is 1,031, which can be added to the prompt within the context length limit for several games. In this case, each demonstration consists of

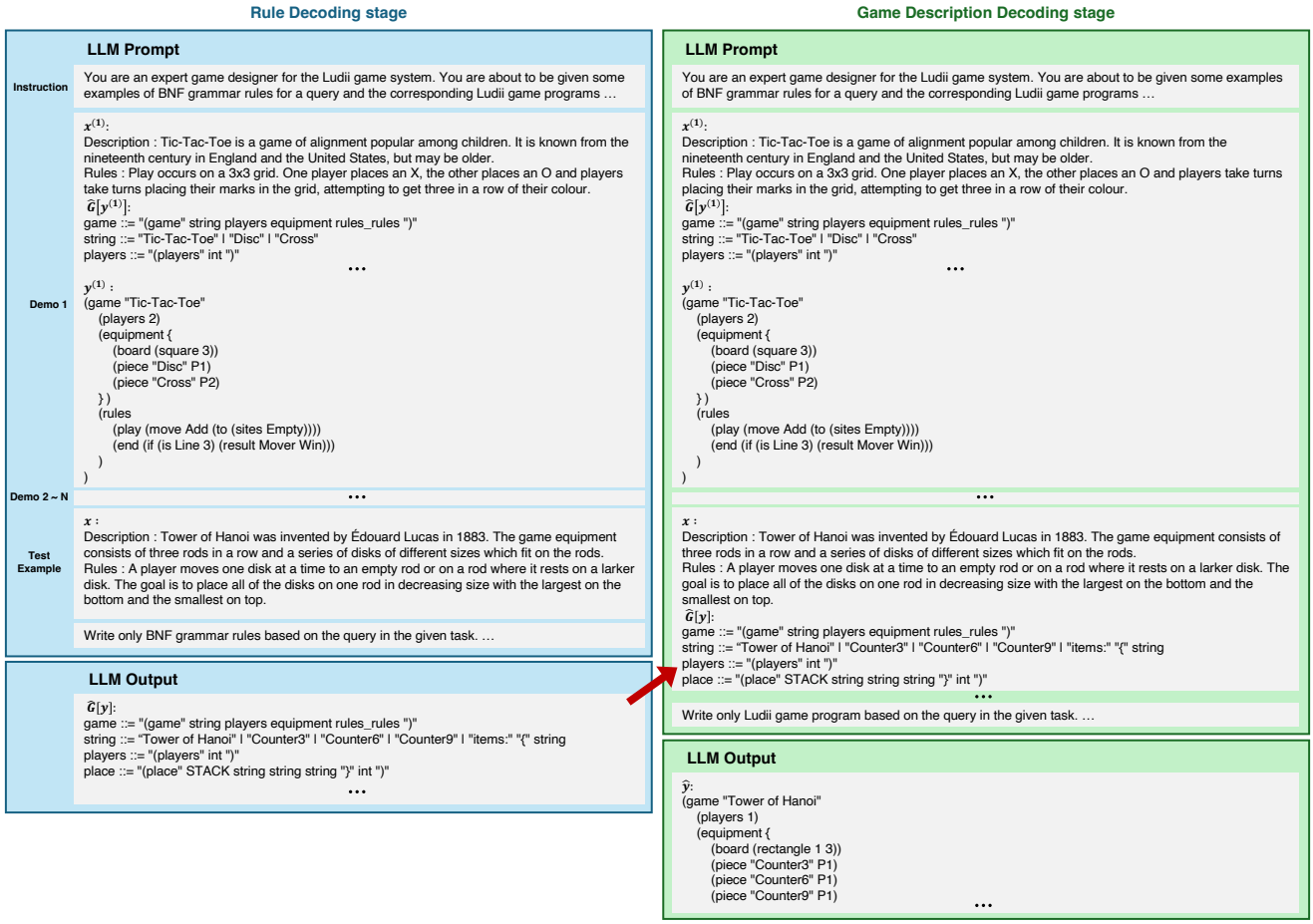

Fig. 4. **An example of grammar-based game description generation result.** We generate game descriptions in two stages: first we generate the required grammar, and then generate the game description based on the grammar. The prompt includes demonstrations for in-context learning and the test example query x, and the demonstration contains the grammar $G[y^{(i)}]$. In the first stage, the minimal grammar $\hat{G}[y]$ that composes y is generated. In the second stage, game description $\hat{y}$ is generated based on the generated $\hat{G}[y]$. The red arrow indicates that $\hat{G}[y]$ generated in the first stage is included in the prompt for the second stage.

$(x^{(i)}, G[y^{(i)}], y^{(i)})$. $G[y^{(i)}]$ is obtained by parsing $y^{(i)}$ with G and collecting the rules necessary to derive $y^{(i)}$.

The generation process consists of two stages. In the first stage, a few demonstration examples and a test example query x are input to the LLM as a prompt to generate the minimal grammar $\hat{G}[y]$ necessary to compose y. The grammar G' that maximizes the following probability is selected as $\hat{G}[y]$,

$$P_{\mathrm{LLM}}(G'|(x^{(i)}, G[y^{(i)}], y^{(i)})_{i=1}^{N}, x). \tag{1}$$

In the second stage, the generated $\hat{G}[y]$ is added to the prompt to generate the game description $\hat{y}$, and the y' that maximizes the following probability is selected as $\hat{y}$:

$$P_{\mathrm{LLM}}(y'|(x^{(i)}, G[y^{(i)}], y^{(i)})_{i=1}^{N}, x, \hat{G}[y]). \tag{2}$$

We show an overview of the two-stage generation process in Fig. 4.

Additionally, in the second stage, although y is conditioned on the grammar $\hat{G}[y]$, the generated game description may not adhere to $\hat{G}[y]$. This is because the LLM just selects the words with the highest likelihood, and does not necessarily comply with the conditions set by the prompt. Similarly, $\hat{G}[y]$ may not be a subset of the original grammar G. Therefore, in the next subsection, we propose decoding methods to improve the consistency of the minimal grammar $\hat{G}[y]$ with the original grammar G, and the consistency of the game description $\hat{y}$ with the grammar $\hat{G}[y]$.

B. Grammar-based Iterative Decoding

In order to overcome issues of inconsistent grammars, we propose decoding methods that iteratively improve the generated grammars. In particular, our decoding methods are divided into two types, one specialized for the grammar $\hat{G}[y]$, and one for the game description $\hat{y}$. When decoding the grammar $\hat{G}[y]$, undefined non-terminal symbols are extracted in one step, and rules to define them are then generated in the next step, in what we call the *Rule Decoding* stage. Similarly, when generating the game description $\hat{y}$, we use the Earley parser [64] to obtain the longest valid continuation that adheres to the grammar $\hat{G}[y]$ in one step, and then complete the remaining parts following the valid continuation in the next step. We refer to this as *Game Description Decoding* stage.

a) Rule Decoding Stage: Our rule decoding stage iteratively applies rule decoding while aiming to ensure that the generated $\hat{G}[y]$ is a subset of the original grammar G.

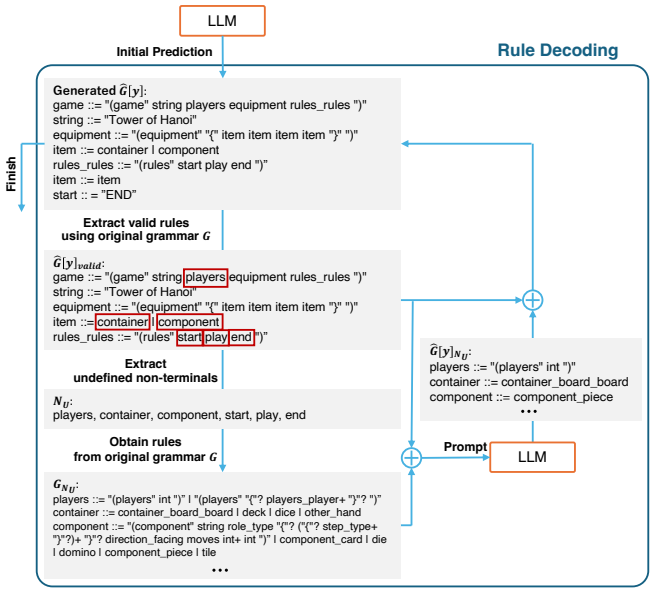

Fig. 5. **Processing flow of Rule Decoding stage.** The Rule Decoding stage starts from the minimal grammar $\hat{G}[y]$ necessary to compose y generated by the LLM, and improves it iteratively. From the grammar $\hat{G}[y]$, the set of rules included in the original grammar G is extracted as $\hat{G}[y]_{\text{valid}}$. Next, undefined non-terminal symbols N_U are extracted from $\hat{G}[y]_{\text{valid}}$. The rules G_{N_U} concerning N_U are obtained from the original grammar G and input to the LLM along with $\hat{G}[y]_{\text{valid}}$. The LLM then generates rules $\hat{G}N_U$ for the undefined non-terminal symbols. Finally, $\hat{G}[y]$ is updated by combining G_{N_U} with $\hat{G}[y]_{\text{valid}}$.

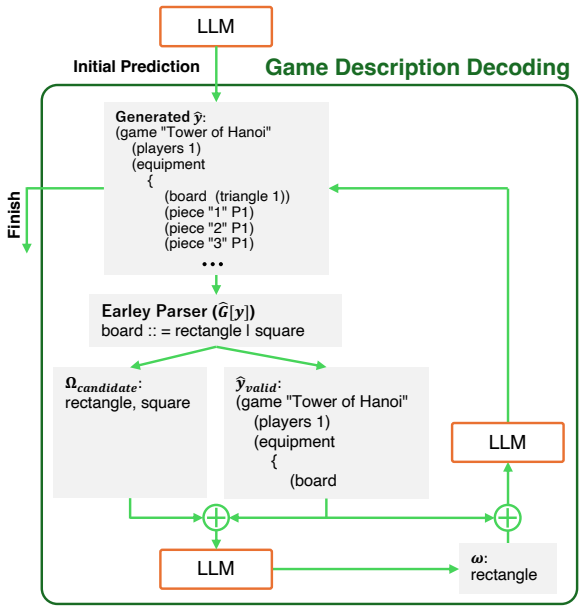

Fig. 6. **Processing flow of Game Description Decoding stage.** The Game Description Decoding stage starts from the initial game description $\hat{y}$ generated by the LLM, and then enhances the results iteratively. Using an Earley parser [64] based on $\hat{G}[y]$, the longest valid subsequence $\hat{y}_{\text{valid}}$ and the subsequent terminal symbol candidates $\Omega_{\text{candidate}}$ are obtained. The LLM then selects the candidate ω from $\Omega_{\text{candidate}}$ that is most suitable as the successor to $\hat{y}_{\text{valid}}$. ω is appended to the end of $\hat{y}_{\text{valid}}$, and the LLM generates the remaining part of the game description $\hat{y}$ that follows. The generated $\hat{y}$ replaces the previous $\hat{y}$ from the earlier step.

In our rule decoding stage, the generation process is multi-step, where each step iteratively improves the grammar $\hat{G}[y]$. The goal of each step is to define the non-terminal symbols that were not defined in the grammar $\hat{G}[y]$ generated in the previous step. We illustrate our rule decoding process in Fig. 5. Initially, $\hat{G}[y]$ is generated using the LLM in the same way as in Eq. (1). From $\hat{G}[y]$, only the valid grammar rules $\hat{G}[y]_{\text{valid}}$ are retained, which are included in the original grammar G. Among $\hat{G}[y]_{\text{valid}}$, the set of undefined non-terminal symbols N_U is extracted. Undefined non-terminal symbols are rules that are used on the right-hand side of rules in $\hat{G}[y]_{\text{valid}}$ but are not defined. They can be automatically and easily extracted by identifying symbols not appearing on the left-hand side of any rules. The set of grammar rules for the undefined non-terminal symbols G_{N_U} is extracted from the original grammar G. G_{N_U} is added to the prompt, and the LLM selects only the necessary rules from G_{N_U} based on the query x. The right-hand side of the rules in G_{N_U} includes options that are unnecessary for constructing y. The role of the LLM is to select the minimum necessary choices to construct y from these options, and to predict the minimal grammar $G[y]_{N_U} = G[y] \setminus \hat{G}[y]_{\text{valid}}$. The rules that maximize the following probability are selected:

$$P_{\text{LLM}}(G'[y]_{N_U} | (x^{(i)}, G[y^{(i)}], y^{(i)})_{i=1}^{N}, x, \hat{G}[y]_{\text{valid}}, G_{N_U}). \tag{3}$$

The generated $\hat{G}[y]_{N_U}$ is combined with $\hat{G}[y]_{\text{valid}}$ to update $\hat{G}[y]$. This process is repeated until there are no more non-terminal symbols or a predetermined limit of updates is reached.

b) Game Description Decoding stage: The game description decoding stage consists of iterative application of game description decoding to generate game descriptions that more accurately adhere to the grammar $\hat{G}[y]$. In each step, it aims to complete the parts of $\hat{y}$ generated in the previous step that cannot be parsed. Initially, $\hat{y}$ is generated using the LLM in the same way as in Eq. (2). An Earley parser [64] is then employed using $\hat{G}[y]$. The Earley parser explores the program from left to right, extracting the longest valid subsequence and the subsequent candidate terminal symbols. Using the generated $\hat{y}$, the Earley parser obtains the valid subsequence $\hat{y}_{\text{valid}}$ and the set of candidate terminal symbols $\Omega_{\text{candidate}}$. Because multiple options for rules are often generated during the rule decoding stage, the LLM selects the optimal terminal symbol ω from $\Omega_{\text{candidate}}$. The ω that maximizes the following probability is then chosen:

$$P_{\text{LLM}}(\omega | (x^{(i)}, G[y^{(i)}], y^{(i)})_{i=1}^{N}, x, \hat{G}[y], \hat{y}_{\text{valid}}, \Omega_{\text{candidate}}). \tag{4}$$

The selected candidate is appended to the end of $\hat{y}_{\text{valid}}$, and the LLM generates the remaining part of the game description $\hat{y}$. Next, the y' that maximizes the following probability is generated:

$$P_{\text{LLM}}(y' | (x^{(i)}, G[y^{(i)}], y^{(i)})_{i=1}^{N}, x, \hat{G}[y], \hat{y}_{\text{valid}} + \omega). \tag{5}$$

The generated $\hat{y}$ updates $\hat{y}$ from the previous step. This process is repeated until the entire generated $\hat{y}$ becomes parsable by the Earley parser or until a predetermined number of updates is reached.

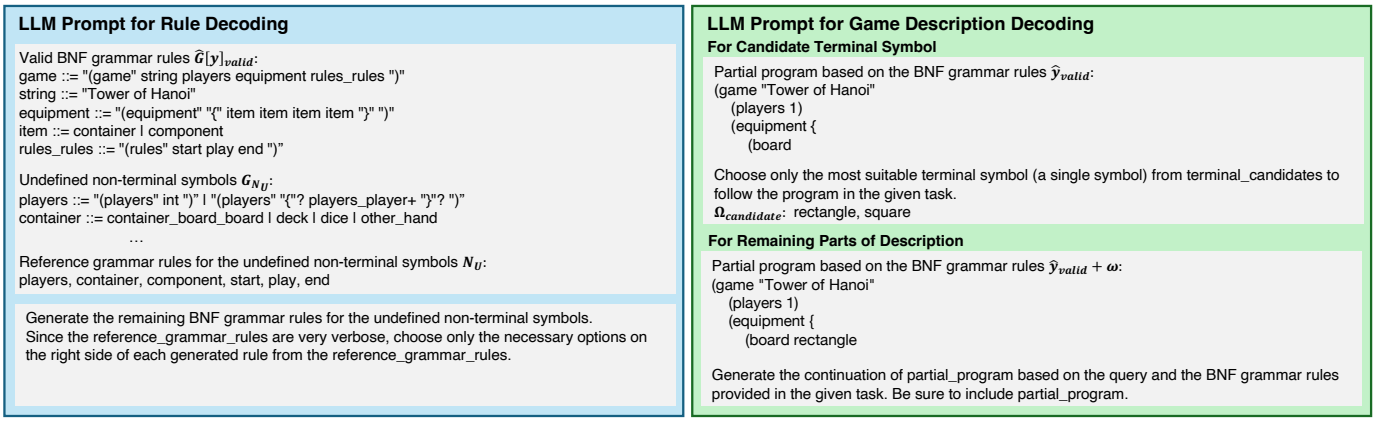

Fig. 7. **Our prompts for grammar-based iterative decoding.**

V. Experiments

A. Datasets

The game descriptions we use for evaluation are obtained from the publicly available Ludii website [65]. In Ludii, each game is assigned a category (*e.g.*, "Tic-Tac-Toe" is categorized under "Board/Space/Line", and "Tower of Hanoi" under "Puzzle/Planning"). From the same category, one test example and three demonstration examples are extracted as a single instance, leveraging the known benefit of using semantically similar examples to improve ICL performance [66]. Note that only examples where the token length of the game description y is 300 or less are used. The token length is calculated using the tokenizer of Llama3-8B-Instruct [9]. The evaluation dataset is constructed from 100 randomly selected instances from all categories, and the same set of instances is used in all evaluations. Each example consists of a natural language instruction x, a Ludii game description y, and the minimal grammar $G[y]$ required to construct y. The instruction x is composed of the metadata "Description" and "Rules" provided by the Ludii game system (see Fig. 1 for an example). To improve the generality of the dataset, we use game descriptions where the unique functions defined within each game are expanded as y. The extended version is instantiated with primitive options and rulesets to avoid using the Ludii game system's meta-language features (definitions, options, rulesets, ranges, constants, etc.). A similar process is adopted in [46]. For more details of the Ludii language, please refer to the Ludii language reference [67]. The grammar $G[y]$ is automatically extracted by the parser using the Lark library [68], which is explained in the next subsection.

B. Methods

We compare the following methods:

- **Random**: A baseline method that generates the grammar $\hat{G}[y]$ necessary for constructing the game description y. Based on $\hat{G}[y]$, the method randomly samples the next expressions following "(game" to generate the game description.
- **Game Description Generation (GDG)**: A baseline method that directly predicts the game description y from the demonstration examples and the query x without using the predicted grammar $\hat{G}[y]$.
- **Grammar-based Game Description Generation (GGDG, ours)**: Our proposed method, which first generates the grammar $\hat{G}[y]$ required to construct the game description y and subsequently generates $\hat{y}$ based on $\hat{G}[y]$. This method employs both rule decoding and game description decoding.
- **SFT+GDG**: A baseline method that predicts the game description from a query using an LLM with supervised fine-tuning (SFT). For SFT training data, pairs of query x and game description y are created from games available on the Ludii portal [65] that are not included in the evaluation dataset.
- **SFT+GGDG (ours)**: A method that combines our proposed GGDG with SFT. For rule decoding, an LLM with SFT applied to query x and grammar $G[y]$ pairs is used. For game description decoding, the same LLM model as in SFT+GDG is used, trained with SFT on query x and game description y pairs.

Since the model acquires knowledge of Ludii through SFT, demonstration examples are not used, *i.e.*, zero-shot inference.

C. Implementation Details

We use a parser built with a Python library called Lark [68] to extract the grammar rules of the Ludii game description. The Ludii grammar in the Lark parser employs the rules listed in the Ludii language reference. In our grammar-based iterative decoding, we set an upper limit on the number of iterations to suppress the number of LLM calls. Based on the ablation study in Sec. VI-C, the iteration limit is set to 20 for rule decoding and 10 for game description decoding. Our prompts for grammar-based iterative decoding are shown in Fig 7.

We use the Llama3-8B-Instruct [9] model as our LLM. We chose an open-source LLM to ensure research reproducibility. Unlike commercial APIs, our approach is not affected by API changes or model updates. Llama3 is the latest series of open-source LLMs provided by Meta. The Llama3 series includes pre-trained models with 8B and 70B parameters. Compared to other models of similar size, the Llama3 series demonstrates superior performance across various benchmarks. To ensure

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

Method	Compilability↑	Functionality↑	ROUGE↑	Normalized Concept Distance↓
Random	1.3±0.7	1.0±0.6	9.8±0.8	0.97±0.02
GDG	27.0±1.2	26.3±0.7	63.5±0.6	0.75±0.01
GGDG (ours)	64.0±1.5	56.7±2.3	60.5±0.6	0.46±0.03
SFT+GDG	59.7±2.4	58.0±0.6	**64.0±0.5**	0.44±0.01
SFT+GGDG (ours)	**72.0±2.1**	**70.3±1.8**	63.8±0.7	**0.33±0.02**

that our framework is practical in environments with limited computational resources, such as local deployments, we select the smaller 8B model. Since the Llama3 models are pre-trained to accept sequences of up to 8,192 tokens, it is necessary to keep the prompts within this context length. To address this, we set the number of demonstration examples to three. We use two NVIDIA RTX 6000 Ada GPUs for our experiments.

For SFT models, we apply SFT to Llama3-8B-Instruct using low-rank adaptation (LoRA) [69]. We set the SFT sequence length to 4096, LoRA alpha and r to 16, and train the model for 3 epochs with a learning rate of 1e-4 and a warmup ratio of 0.03.

VI. EXPERIMENTAL RESULTS

A. Evaluation Metrics

To evaluate the generated game descriptions, we use the following metrics:

- **Compilability**: The proportion of games that can be parsed and compiled by the Ludii game engine. If a game cannot be compiled, it is not evaluated in the functionality metric. The score is normalized from 0 to 100.
- **Functionality**: The proportion of playable games. Although a game may compile without errors, it is still considered non-functional if conditions make it unplayable—such as when piece movements rely on undefined positions. The score is normalized from 0 to 100.
- **ROUGE** [70]: This is an evaluation metric commonly used in program synthesis to measure the degree of linguistic match between the generated game description and the ground truth game description [71]. It ranges from 0 to 100, with higher values indicating a greater degree of match. The calculation of this metric does not consider syntactic correctness but instead focuses on the similarity between texts. We use the F1 score of ROUGE-L. This value is calculated for each piece of test data, and the average of all data is reported.
- **Normalized Concept Distance (NCD)**: NCD measures how closely a predicted game matches its ground truth in Ludii. Following [16], [18], games are represented as concept-value vectors derived from semantic features and from behavioral data obtained through automated playouts using a random policy. For example, these values include attributes such as the proportion of turns with at least one legal move and the proportion of the board used at least once. The cosine distance between these vectors gives NCD. Similar to [46], we run 50 playouts for the ground truth and 10 for the predicted games due to computational costs. Since non-functional games cannot compute their concept distance, their distance is set to 1.0. NCD is averaged over all test data, serving as a quality measure for game description generation.

We conducted experiments with 3 different seeds, and each value in the results shows the mean and standard error across the seeds.

B. Comparison with Baseline Methods

Table II shows the comparison results with baseline methods. GGDG outperforms GDG with Compilability +37.0, Functionality +30.4, and NCD -0.29, demonstrating that our rule decoding and game description decoding effectively improve the grammatical accuracy of generated game descriptions. In ROUGE, GGDG scores -3.0 lower than GDG. Since ROUGE evaluates the level of reproduction in linguistic expressions, it may undervalue different description methods generated by GGDG that provide the same playable experience. For example, the ending rule for Tic-tac-toe can be defined as either "win by aligning three pieces" or "lose when the opponent aligns three pieces" – while these expressions have the same meaning as rules, they would result in a lower ROUGE score.

While GGDG shows a +4.3 advantage in Compilability compared to SFT+GDG, it performs worse in Functionality (-1.3), ROUGE (-3.5), and NCD (-0.02). This suggests that although GGDG can generate grammatically accurate game descriptions, it still faces challenges compared to SFT in satisfying functionality requirements and reproducing the game quality of the ground truth. SFT+GGDG achieves the best scores across Compilability (72.0), Functionality (70.3), and NCD (0.33) metrics, demonstrating that SFT has a complementary effect on GGDG.

Regarding the baseline results, random generation methods can hardly produce grammatically valid game descriptions. While GDG shows high ROUGE scores, it performs poorly on other metrics. This suggests that although LLMs alone can generate game descriptions that are superficially similar in linguistic expression, without accessing Ludii's grammar through our decoding methods, they struggle to generate grammatically and functionally accurate game descriptions. Additionally, SFT+GDG shows improved performance across all metrics compared to GDG, indicating that providing the model with Ludii knowledge through SFT further enhances grammatical accuracy.

TABLE III
ABLATION STUDY OF OUR PROPOSED METHOD. "ORACLE GRAMMAR" INDICATES CASES WHERE GRAMMAR $G[y]$ EXTRACTED FROM GAME DESCRIPTION y IS USED INSTEAD OF GRAMMAR $\hat{G}[y]$ PREDICTED BY THE LLM. "GRAMMAR" INDICATES THE INCLUSION OF GRAMMAR IN THE QUERY DURING GAME DESCRIPTION GENERATION. THE BEST RESULTS ARE IN **BOLD**.

Method	Grammar	Rule Decoding	Game Description Decoding	Compilability↑	Functionality↑	ROUGE↑	Normalized Concept Distance↓
GDG				27.0±1.2	26.3±0.7	63.5±0.6	0.75±0.01
GGDG w/o RD, GDD	✓			48.3±0.3	43.3±1.2	61.6±0.1	0.59±0.01
GGDG w/o GDD	✓	✓		52.0±3.6	47.3±2.3	61.3±0.2	0.55±0.02
GGDG	✓	✓	✓	**64.0±1.5**	56.7±2.3	60.5±0.6	**0.46±0.03**
Oracle Grammar							
GGDG w/o RD, GDD	✓			54.0±1.7	49.3±0.7	61.7±0.2	0.53±0.01
GGDG w/o RD	✓		✓	62.0±1.5	**57.3±2.2**	**64.5±0.5**	**0.46±0.02**

TABLE IV
COMPARISON OF THE ITERATION LIMITS FOR RULE DECODING. GAME DESCRIPTION DECODING IS NOT USED (I.E., GGDG W/O GDD). THE BEST RESULTS ARE IN **BOLD**.

Iteration	Compilability↑	Functionality↑	ROUGE↑	NCD↓
10	50.0±1.0	44.7±2.0	61.5±0.1	0.58±0.02
20	**51.3±3.5**	**46.7±2.4**	61.3±0.2	**0.55±0.02**
30	47.7±2.7	44.0±3.5	**62.1±0.2**	0.58±0.03

TABLE V
COMPARISON OF THE ITERATION LIMITS FOR GAME DESCRIPTION DECODING. THE NUMBER OF ITERATIONS FOR RULE DECODING IS FIXED AT 20. THE BEST RESULTS ARE IN **BOLD**.

Iteration	Compilability↑	Functionality↑	ROUGE↑	NCD↓
5	60.0±3.2	53.7±2.8	**61.1±0.2**	0.50±0.02
10	64.0±1.5	56.7±2.3	60.5±0.6	**0.46±0.03**
20	**64.7±0.3**	**57.3±1.9**	60.6±0.2	0.47±0.01

TABLE VI
COMPARISON OF GAME CATEGORIES IN DEMONSTRATION EXAMPLES. "SAME" CATEGORY INDICATES THAT EXAMPLES ARE FROM THE SAME CATEGORY AS THE TEST INSTANCE, WHILE "CROSS" CATEGORY INDICATES THAT THEY ARE FROM DIFFERENT CATEGORIES. THE BEST RESULTS ARE IN **BOLD**.

Method	Compilability↑	Functionality↑	ROUGE↑	NCD↓
Cross Category				
GDG	5.3±0.3	3.7±0.7	46.0±0.0	0.97±0.00
GGDG	36.0±0.6	24.6±0.3	42.3±0.1	0.79±0.00
Same Category				
GDG	27.0±1.2	26.3±0.7	**63.5±0.6**	0.75±0.01
GGDG	**64.0±1.5**	**56.7±2.3**	60.5±0.6	**0.46±0.03**

C. Ablation Study

GGDG Components. We conduct an ablation study on GGDG and summarize the results in Tab. III. We confirm that adding more techniques improves the scores for Compilability, Functionality, and NCD. Among the improvements from GDG to GGDG, grammar accounts for the largest share, comprising 57.6% of Compilability, 55.9% of Functionality, and 55.2% of NCD improvements. This is followed by game description decoding, which accounts for 32.4% of Compilability, 30.9% of Functionality, and 31.0% of NCD improvements, demonstrating their substantial impact. Furthermore, the improvement in Functionality and ROUGE scores when using oracle grammar suggests that there is still room for improvement in grammar generation through rule decoding.

Iteration Limit for Decoding. We investigate the impact of iteration limits in our decoding approach. First, Tab. IV shows the results of rule decoding. When the iteration limit is set to 30, we find that the Compilability and Functionality scores are the lowest. This may be because increasing the iteration limit leads to the inclusion of unnecessary rules, potentially degrading performance. When the iteration limit is set to 20, Compilability, Functionality, and NCD scores show the best results. Therefore, we set the iteration limit to 20 for subsequent experiments.

Tab. V shows the results of game description decoding. When the iteration limit is set to 5, we find that the Compilability and Functionality scores are the lowest. This is likely due to insufficient iterations for improving the game description. When the iteration limit is set to 10 or 20, we observe minimal differences across most metrics. When the iteration limit is 10, NCD is lowest at 0.46 and the computational cost is also low, therefore we set the iteration limit to 10 in other experiments.

Game Category of Demonstration Examples. We investigate how the category of demonstration examples affects game performance. In our proposed method, we use instances from the same category as the test instances for demonstration examples (Same Category). We compare this with using instances from different categories (Cross Category), and summarize the results in Tab. VI. In Cross Category, demonstration examples are randomly selected from all games except for the category of the test instance. We find that obtaining demonstration examples from the same category as test instances significantly improves performance across all evaluation metrics.

D. Impact of Game Characteristics and Model Configurations

Game Description Length. We investigated the impact of game description length on performance. We compared three groups based on token length: 0-300, 300-500, and 500-1,000. Token lengths were calculated using the Llama-3-8B-Instruct tokenizer. The results are summarized in Tab. VII. For 0-300 and 300-500 tokens, GGDG outperforms GDG in Compilability, Functionality, and NCD metrics, demonstrating its ability to generate more grammatically accurate game descriptions. Meanwhile, GGDG showed declining performance across all

TABLE VII
COMPARISON OF TEST GAME LENGTHS. *300 - 500* INDICATES THAT THE EVALUATION IS PERFORMED ON GAMES WITH GAME DESCRIPTION TOKEN LENGTHS BETWEEN 300 AND 500.

Method	Compilability↑	Functionality↑	ROUGE↑	NCD↓
0 - 300				
GDG	27.0±1.2	26.3±0.7	63.5±0.6	0.75±0.01
GGDG	64.0±1.5	56.7±2.3	60.5±0.6	0.46±0.03
300 - 500				
GDG	41.0±2.1	39.0±1.0	59.7±0.2	0.68±0.01
GGDG	54.3±2.2	48.7±1.3	59.4±0.5	0.59±0.01
500 - 1000				
GDG	36.0±2.3	32.7±2.6	49.9±0.3	0.74±0.02
GGDG	33.3±1.2	29.7±0.9	47.3±0.6	0.77±0.00

TABLE VIII
COMPARISON OF TEST INSTANCE CATEGORIES.

Method	Compilability↑	Functionality↑	ROUGE↑	NCD↓
board/race				
GDG	10.0±5.8	10.0±5.8	58.2±0.2	0.90±0.06
GGDG	60.0±5.8	60.0±5.8	58.8±0.2	0.42±0.05
board/sow				
GDG	38.9±5.6	38.9±5.6	78.8±1.0	0.75±0.06
GGDG	55.6±5.6	55.6±5.6	81.3±0.9	0.51±0.06
puzzle				
GDG	7.4±1.9	7.4±1.9	51.3±0.5	0.98±0.02
GGDG	61.1±6.4	59.3±6.7	52.1±0.7	0.57±0.12
board/space/line				
GDG	26.9±0.9	24.4±0.5	64.2±0.3	0.78±0.00
GGDG	68.7±0.9	63.7±1.3	62.4±0.4	0.42±0.02
board/war				
GDG	57.7±4.4	57.7±4.4	71.0±0.1	0.45±0.04
GGDG	71.8±7.1	62.8±9.0	71.8±0.3	0.41±0.09

metrics as token length increased, dropping from 64.0 to 33.3, Functionality from 56.7 to 29.7, ROUGE from 60.5 to 47.3, and NCD worsening from 0.46 to 0.77. For 500-1,000 tokens, GDG slightly outperformed GGDG across all metrics. These results suggest that GGDG struggles with long game descriptions. We discuss this limitation in Sec VII.

Game Category of Test Games. We investigate the impact of game categories of test instances. We compare GDG and GGDG across 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 demonstration examples are from the same category as the test instances. We use instances with game description token lengths of 300 or less as test games.

The results are summarized in Tab. VIII. Across all categories, our proposed GGDG outperforms GDG in terms of Compilability, Functionality, and NCD metrics, which is consistent with our previous experimental results. For ROUGE scores, while the superior method varies by category, the difference between GDG and GGDG remains within 2 points across most categories.

In comparisons across categories, GDG's performance shows significant variation, with the puzzle category showing the lowest scores (Compilability/Functionality: 7.4, NCD: 0.98) for GDG and the largest improvement margin (+53.7/+51.9 and -0.41 respectively) when using GGDG. Conversely, the board/war categories showed GDG's highest performance levels (Compilability/Functionality: 57.7, NCD: 0.45) and the smallest margin of improvement with GGDG (+14.1/+5.1 and -0.04 respectively). This variation may be attributed to games in the puzzle category having more complex logic compared to other categories, making it difficult for LLMs to learn grammatically correct patterns from demonstration examples, or possibly due to limited knowledge about puzzle category games obtained during pre-training. Meanwhile, GGDG showed smaller performance variations between categories compared to GDG, with scores ranging from 55.6 to 71.8 for Compilability, 55.6 to 63.7 for Functionality, and 0.41 to 0.57 for NCD, demonstrating more stable performance.

TABLE IX
COMPARISON OF LLM MODELS. THE BEST RESULTS ARE IN **BOLD**.

Method	Compilability↑	Functionality↑	ROUGE↑	NCD↓
Llama-3.2-3B-Instruct				
GDG	18.7±0.7	17.3±0.3	59.2±0.3	0.83±0.00
GGDG	28.0±1.5	25.7±2.0	55.4±0.1	0.76±0.02
Llama-3-8B-Instruct				
GDG	27.0±1.2	26.3±0.7	63.5±0.6	0.75±0.01
GGDG	64.0±1.5	56.7±2.3	60.5±0.6	0.46±0.03
gpt-4o				
GDG	27.7±0.3	27.7±0.3	67.7±0.1	0.74±0.00
GGDG	**70.7±0.9**	**59.3±0.9**	**68.2±0.0**	**0.44±0.01**

LLM Models. We compare the performance of different LLM models. We primarily used Llama-3-8B-Instruct in our experiments and compare its performance to that of Llama-3.2-3B-Instruct, a smaller LLM model, and gpt-4o, a more powerful LLM model. The results are summarized in Tab. IX. GGDG improves Compilability, Functionality, and NCD scores over GDG across all models. With gpt-4o, GGDG also improves the ROUGE score. Comparing across models, the 8B model shows significant improvements in all evaluation metrics compared to the 3B model. For example, with GGDG, upgrading from the 3B model to the 8B model improves Compilability by +36.0, Functionality by +31.0, ROUGE by +5.1, and NCD by -0.30. gpt-4o achieves the highest scores across all evaluation metrics. The improvements over the 8B model are +6.7 in Compilability, +2.6 in Functionality, +7.7 in ROUGE, and -0.02 in NCD. We observe that the improvements in Compilability, Functionality, and NCD are larger when moving from the 3B model to the 8B model compared to moving from the 8B model to gpt-4o. This suggests that while increasing model size brings significant performance improvements, the room for improvement becomes limited between already high-performing models. Therefore, if higher performance is desired, selecting powerful models like gpt-4o is effective. In contrast, if there are resource constraints, such as API costs or computational resource limitations, the 8B model can be considered a balanced choice between performance and cost.

SFT+GDG
Not Compilable

```
1 (game "Latin Square"
2    (players 1)
3    (equipment
4       {
5          (board (square 4) (values Cell (range 1 4)))
6          (regions { Columns Rows })
7       }
8    )
9    (rules
10      (play (satisfy {(forAll Region (== 0 (state at:(to))))}))
11      (end (if (is Solved) (result P1 Win)))
12   )
13 )
```

GGDG
Functional

```
1 (game "Latin Square"
2    (players 1)
3    (equipment
4       {
5          (board (square 5) (values Cell (range 1 5)))
6          (regions { Columns Rows })
7       }
8    )
9    (rules
10      (play (satisfy {(all Different except:0)} ))
11      (end (if (is Solved) (result P1 Win)))
12   )
13 )
```

SFT+GGDG
Functional

```
1 (game "Latin Square"
2    (players 1)
3    (equipment
4       {
5          (board (square 4) (values Cell (range 1 4)) )
6          (regions { Columns Rows })
7       }
8    )
9    (rules
10      (play (satisfy { (all Different) (is Full) }))
11      (end (if (is Solved) (result P1 Win)))
12   )
13 )
```

Ground Truth

```
1 (game "Latin Square"
2    (players 1)
3    (equipment
4       {
5          (board (square 5) (values Cell (range 1 5)))
6          (regions { Columns Rows })
7       }
8    )
9    (rules
10      (play (satisfy (all Different)))
11      (end (if (is Solved) (result P1 Win)))
12   )
13 )
```

Fig. 8. **Comparison of generation results with baseline methods for Latin Square.** Since the board size is not specified in the query, it varies depending on the generated results. The ground truth is set to size 5 as an example.

E. Qualitative Results

Comparison with Baseline Methods. We conduct a qualitative analysis comparing SFT+GDG, which is the most promising among the baseline methods, with our proposed methods, GGDG and SFT+GGDG. Figure 8 shows the generation results for Latin Square. Latin Square is a puzzle where players place numbers in an n×n grid such that the same number does not repeat in each row and column.

The result of SFT+GDG is neither compilable nor functional due to the part shown in the red frame: "(forAll Region (== 0 (state at:(to)))". This issue stems mainly from two grammatical problems. First, while the Ludii grammar defines that a "forAll" clause must be followed by a "puzzleElementType" (either "Cell", "Edge", "Vertex", or "Hint"), an undefined terminal symbol, "Region", is used. Second, the expression "(== 0 (state at:(to)))" cannot be parsed because the operator "==" is not defined in the Ludii grammar.

In contrast, GGDG's result is both compilable and functional, generating a game description almost identical to the ground truth. In particular, it shows significant improvement in that the result contains no grammatical errors. However, in the blue frame, "except:0" (a constraint requiring all values to be different except for index 0) is added after "all Different", and this rule is not included in the game rules of Latin Square.

Furthermore, the SFT+GGDG result shows more improvement and is consistent with the Latin Square game rules. The "is Full" rule, which requires all cells to be filled, is redundant as it is already included in the content of the "all Different" rule, but it does not contradict the Latin Square rules. These results suggest that combining SFT and GGDG improves performance.

GGDG Components. We investigate the effects of GGDG components through qualitative analysis. Figure 9 shows the generation results for Tic-Tac-Toe. The GDG results are not compilable due to the "count Pips" shown in the red frame. The appearance of the term "Pips" occurs because the LLM is influenced by the dice in the Tic-Tac-Die game from the demonstration examples. Tic-Tac-Die is explained as follows: "Tic-Tac-Die is played similarly to Tic-Tac-Toe except that players roll a D9 dice each turn to dictate where they move (dice pips show the cell index to move to)."

While the pips disappear in the results of GGDG w/o RD, GDD, they are still not compilable. This is due to the extra clause about "PASS" in the blue frame. This occurs because the predicted "move" grammar contains a "PASS" clause not included in the ground truth.

The grammar results of GGDG w/o GDD show that by adding rule decoding, they match the ground truth grammar. The generated game description is functional. However, this description includes "do," which is not in the predicted grammar. The presence of this term, which appears in the Tic-Tac-Die demonstration example, suggests that the LLM is mimicking it.

In the GGDG results, adding game description decoding reduces the occurrence of expressions not defined in the grammar, such as "do". This game description is functional and closest to the ground truth among the compared methods. However, since the condition "(is line 3)" is applied equally to player1 and player2 as shown in the purple frame, it cannot accurately determine which player has won. This suggests there is still room for improvement in GGDG.

Analysis of Failure Cases. We examine the analysis results of Bara Guti (Bihar) as an example of a failure case, as shown in Fig. 10. This game uses a board that has three concentric circles, with four diameters dividing it into eight equal sections, with counters placed in each section. In analyzing the generated results, we note that "Markers" and "Counters" refer to the same type of piece.

While SFT+GDG generated compilable output, it did not produce functional results due to the parts highlighted in red in Fig. 10. "concentric 1 8 8" specifies the number of cells in each concentric ring sequentially, defining a total of 17 cells. However, this description becomes non-functional because lines 19 and 21 specify marker initial positions with indices of 17 or greater. Additionally, the specified initial positions differ from the ground truth.

In contrast, while GGDG's generated results demonstrate functionality, the definitions of the board and counter initial positions differ from the ground truth. Specifically, it generates a board by connecting two triangles on lines 19 and 22, and places counters at positions expanded two steps from the bottom using the expression "expand (sites Bottom) steps:2" (highlighted in blue). This difference is presumed to be influenced by the existing demonstration example of Lau Kata Kati: "(merge (wedge 4) (shift 0 3 (rotate 180 (wedge 4))))".

Finally, while SFT+GGDG's results maintain functionality and return to the concentric circle board structure, the number of concentric circles, cell count, and Marker initial positions are incorrectly predicted, highlighted in green. These limita-

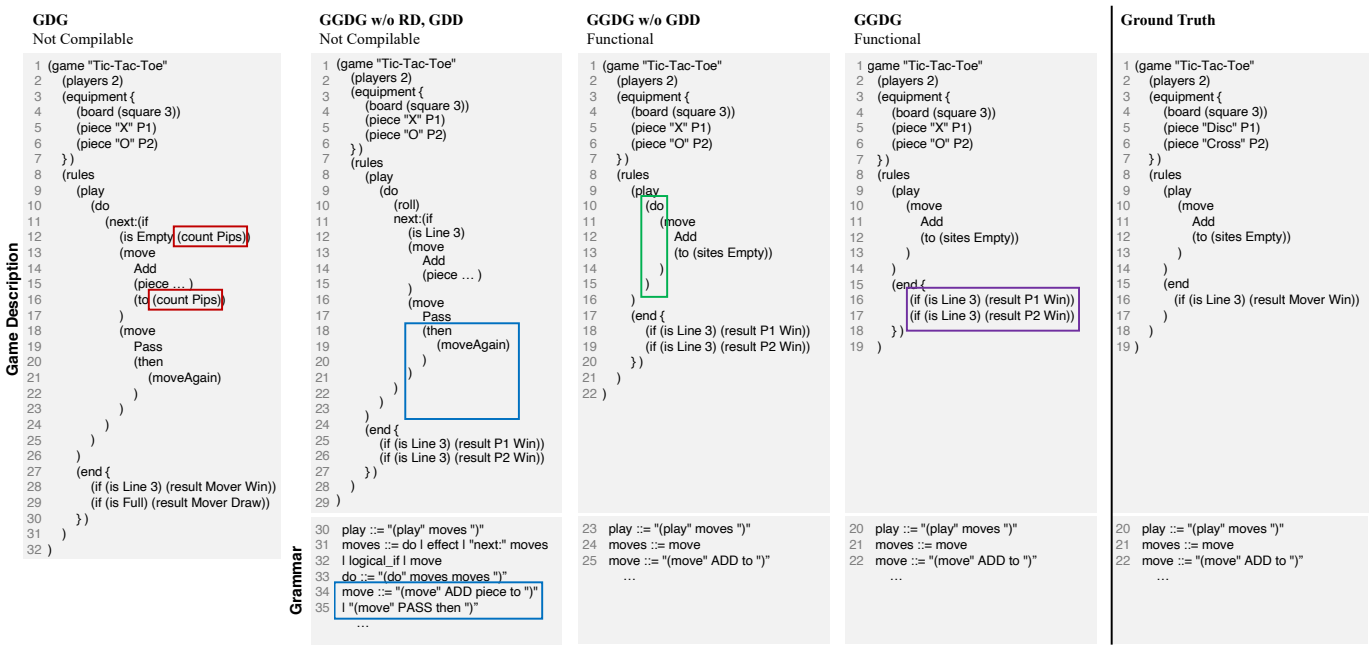

Fig. 9. **Impact of GGDG components on generation results for Tic-Tac-Toe.** Components are added from left to right, with the ground truth on the far right. The middle row shows the game description, and the bottom row shows the results of grammar generation for only the methods that performed grammar generation. Since the grammar results are highly redundant, only the important parts are shown.

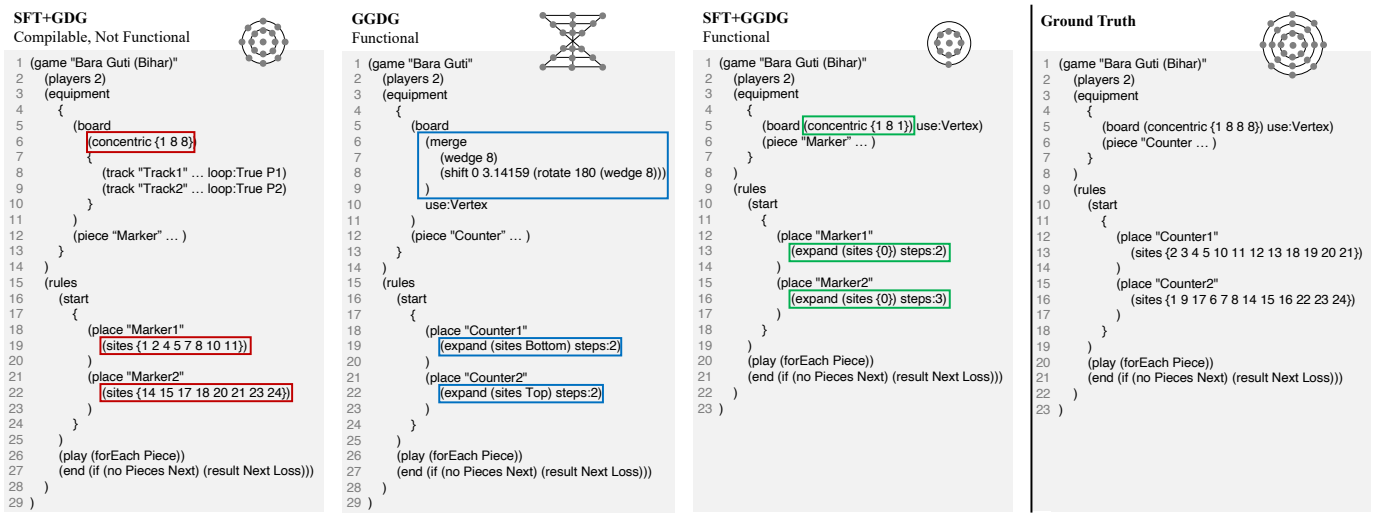

Fig. 10. **Comparison of generation results for Bara Guti (Bihar) as a failure example.** Some parts of the game description are omitted due to space constraints. The board appears in the upper right of each game description. Due to space constraints, GGDG's results display a wedge of 4 instead of 8.

tions can be attributed to LLM's lack of ability to understand spatial structures like boards [72]. We discuss these limitations of spatial understanding in Sec VII.

VII. Limitations and Discussion

As shown in Tab. VII, GGDG's performance decreases with longer game descriptions. It is known that LLM performance degrades with longer input sequences [73]. We believe that GGDG struggles with processing longer game descriptions because its input token length is increased by the grammar, compared to GDG. The development of LLMs capable of handling longer contexts is ongoing [74], [75], and newer LLM models may help mitigate this issue.

As shown in Fig. 10, it is difficult to compensate for LLMs' lack of spatial understanding capabilities through SFT. It has been found that LLMs with 70B parameters demonstrate superior capabilities compared to models with 7B or 13B parameters in this regard [72]. Using a larger LLM, such as the 70B parameter models, is a possible option to increase such capabilities. However, it should be noted that SFT for large-parameter LLMs requires substantial computational resources, which can be problematic in other ways.

Our proposed framework may have issues with inference time and cost because it repeats LLM inference multiple times. Specifically, the inference time per game averages 7.7 seconds for GDG and 143.9 seconds for GGDG. This issue can

be mitigated by caching LLM responses, setting appropriate limits on the number of LLM inferences, and improving the prompts. Additionally, given recent advances in inference acceleration techniques [76], these constraints are expected to be lessened as newer techniques are developed.

The evaluation in this paper relies solely on automatically calculable metrics and does not include human evaluations. We believe that human evaluation could capture errors that automated evaluation cannot fully cover, potentially leading to further improvements and leads for future research directions.

VIII. Broader Impact

The use of LLMs in the field of video games is associated with ethical issues related to sustainability, copyright, explainability, and biases [39]. In this section, we discuss the problems and their mitigation strategies within our proposed framework from these perspectives.

Our research raises concerns about the carbon footprint of LLMs. Our framework does not include training LLMs, and only LLMs' inference impacts the environment. This issue can be mitigated by reducing the number of inference calls through caching LLM responses and providing more effective demonstrations. Additionally, using models that offer enhanced performance for the same computational load can further reduce environmental impact.

Our research is related to copyright issues concerning input and output data. It is common practice for LLMs to be trained using copyrighted data [39]. Since our framework does not include the training process, it does not directly cause this issue. However, users should consider the training data of LLMs when employing our framework. Our approach automatically generates game descriptions without human intervention, and these outputs may not be copyrighted. Developing our framework into an interactive approach with human designers may allow the final output to qualify for copyright.

The generation process of LLMs is opaque, and there are studies examining whether LLMs' black-box nature hinders the transparency of PCG [44]. In our research as well, the process of each LLM call remains similarly unclear. However, our grammar-based iterative decoding method breaks down the generation process into multiple steps, which should enhance our understanding of it.

LLMs are trained on data collected from the internet, which introduces biases. Since game design tends to reflect the culture of its time and region, these biases negatively impact the design process when using LLMs. Our framework could potentially mitigate the issue by incorporating examples from various eras and regions in the demonstrations for LLMs. Ludii collects historically influential games from diverse regions and provides metadata on the regions where each game is rooted, which can contribute to mitigating this problem.

IX. Conclusions

We have proposed a novel framework that combines large language models with game description languages for generating game descriptions from text. Our approach integrates GDL grammar into the generation process, enabling the creation of structurally coherent game descriptions. By introducing iterative refinement decoding methods specialized for both grammar generation and game description generation, we have seen improvements in the grammatical accuracy of game descriptions. Extensive experimental results demonstrate that our framework is effective in improving grammatical accuracy within game description generation. Future research may explore fine-tuning specialized LLMs for each subtask of rule and game description completion performed at each step of iterative refinement decoding, as well as the utilization of more efficient and powerful LLMs. Additionally, it would be valuable to qualitatively validate the findings of our approach beyond the Ludii dataset through conducting user studies with actual humans.

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.
[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] "Introducing meta llama 3: The most capable openly available llm to date," 2024.
[10] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," 2023.
[11] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, "Program synthesis with large language models," 2021.
[12] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proc. of ACM SIGPLAN*, 2022.
[13] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Proc. of NeurIPS*, 2020.
[14] C. Hu, Y. Zhao, and J. Liu, "Game generation via large language models," 2024.
[15] F. Essalmi and L. Ayed, "Graphical uml view from extended backus-naur form grammars," in *Proc. of ICALT*, 2006.
[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] E. Piette, M. Stephenson, D. J. Soemers, and C. Browne, "General board game concepts," in *Proc. of CoG*, 2021.
[19] 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.
[20] D. J. N. J. Soemers, Éric Piette, M. Stephenson, and C. Browne, "The ludii game description language is universal," 2024.
[21] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *IEEE Symposium On Computational Intelligence and Games*, 2008.

[22] C. Browne and F. Maire, "Evolutionary game design," *IEEE Trans. Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[23] M. P. Eladhari, A. Sullivan, G. Smith, and J. Mccoy, "Ai-based game design : Enabling new playable experiences," 2011.

[24] M. Treanor, A. Zook, M. P. Eladhari, J. Togelius, G. Smith, M. Cook, T. Thompson, B. Magerko, J. Levine, and A. Smith, "Ai-based game design patterns," in *Proc. of FDG*, 2015.

[25] A. Sarkar and S. Cooper, "Towards game design via creative machine learning (gdcml)," in *Proc. of CoG*, 2020.

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

[27] 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.

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

[29] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *IEEE Trans. Games.*, vol. 10, no. 3, pp. 257–270, 2018.

[30] J. Liu, S. Snodgrass, A. Khalifa, S. Risi, G. N. Yannakakis, and J. Togelius, "Deep learning for procedural content generation," *Neural Computing and Applications*, vol. 33, no. 1, pp. 19–37, 2021.

[31] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," in *Proc.of AIIDE*, 2020.

[32] M. Cook, S. Colton, and J. Gow, "The angelina videogame design system—part i," *IEEE Trans. Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, 2017.

[33] A. Liapis, G. N. Yannakakis, M. J. Nelson, M. Preuss, and R. Bidarra, "Orchestrating game generation," *IEEE Trans. Games.*, vol. 11, no. 1, pp. 48–68, 2019.

[34] A. Summerville, C. Martens, B. Samuel, J. Osborn, N. Wardrip-Fruin, and M. Mateas, "Gemini: Bidirectional generation and analysis of games via asp," in *Proc. of AAAI*, 2018.

[35] M. Guzdial and M. O. Riedl, "Conceptual game expansion," *IEEE Trans. Games.*, vol. 14, no. 1, pp. 93–106, 2022.

[36] M. U. Nasir, S. James, and J. Togelius, "Word2world: Generating stories and worlds through large language models," 2024.

[37] T. Machado, D. Gopstein, A. Nealen, O. Nov, and J. Togelius, "Ai-assisted game debugging with cicero," in *Proc. of CEC*, 2018.

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

[39] 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.

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

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

[42] 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.

[43] 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.

[44] A. Moradi Karkaj, M. J. Nelson, I. Koutis, and A. K. Hoover, "Prompt wrangling: On replication and generalization in large language models for pcg levels," in *Proc. of FDG*, 2024.

[45] A. Anjum, Y. Li, N. Law, M. Charity, and J. Togelius, "The ink splotch effect: A case study on chatgpt as a co-creative game designer," in *Proc. of FDG*, 2024.

[46] 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.

[47] R. Shin, C. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. Van Durme, "Constrained language models yield few-shot semantic parsers," in *Proc. of EMNLP*, 2021.

[48] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing incrementally for constrained auto-regressive decoding from language models," in *Proc. of EMNLP*, 2021.

[49] G. Poesia, A. Polozov, V. Le, A. Tiwari, G. Soares, C. Meek, and S. Gulwani, "Synchromesh: Reliable code generation from pre-trained language models," in *Proc. of ICLR*, 2022.

[50] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-constrained decoding for structured NLP tasks without finetuning," in *Proc. of EMNLP*, 2023.

[51] K. Park, J. Wang, T. Berg-Kirkpatrick, N. Polikarpova, and L. D'Antoni, "Grammar-aligned decoding," 2024.

[52] S. Ugare, T. Suresh, H. Kang, S. Misailovic, and G. Singh, "Improving llm code generation with grammar augmentation," 2024.

[53] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley, "Evolution through large models," in *Handbook of Evolutionary Machine Learning*. Springer, 2023, pp. 331–366.

[54] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," in *Proc. of ICLR*, 2024.

[55] P. Ma, T.-H. Wang, M. Guo, Z. Sun, J. B. Tenenbaum, D. Rus, C. Gan, and W. Matusik, "Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery," in *Proc. of ICML*, 2024.

[56] E. Meyerson, M. J. Nelson, H. Bradley, A. Gaier, A. Moradi, A. K. Hoover, and J. Lehman, "Language model crossover: Variation through few-shot prompting," *ACM Trans. Evol. Learn. Optim.*, vol. 4, no. 4, 2024.

[57] M. U. Nasir, S. Earle, J. Togelius, S. James, and C. Cleghorn, "Llmatic: Neural architecture search via large language models and quality diversity optimization," in *Proc. of GECCO*, 2024.

[58] B. Wang, Z. Wang, X. Wang, Y. Cao, R. A. Saurous, and Y. Kim, "Grammar prompting for domain-specific language generation with large language models," in *Proc. of NeurIPS*, 2023.

[59] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. of NeurIPS*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022.

[60] J. Andreas, J. Bufe, D. Burkett, C. Chen, J. Clausman, J. Crawford, K. Crim, J. DeLoach, L. Dorner, J. Eisner *et al.*, "Task-oriented dialogue as dataflow synthesis," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 556–571, 2020.

[61] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proc. of national conference on artificial intelligence*, 1996.

[62] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "PAL: Program-aided language models," in *Proc. of ICML*, 2023.

[63] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," in *Proc. of NeurIPS*, 2022.

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

[65] Digital Ludeme Project, "Ludii portal." [Online]. Available: https://ludii.games/index.php

[66] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, "What makes good in-context examples for GPT-3?" in *Proc. of ACLW*, 2022.

[67] Digital Ludeme Project, "Ludii language reference." [Online]. Available: https://ludii.games/downloads/LudiiLanguageReference.pdf

[68] ——, "Lark parser." [Online]. Available: https://github.com/lark-parser/lark

[69] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Proc. of ICLR*, 2022.

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

[71] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," 2024.

[72] Y. Yamada, Y. Bao, A. K. Lampinen, J. Kasai, and I. Yildirim, "Evaluating spatial understanding of large language models," *JMLR Trans. Machine Learning Research*, 2024.

[73] T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen, "Long-context llms struggle with long in-context learning," 2024.

[74] H. Jin, X. Han, J. Yang, Z. Jiang, Z. Liu, C.-Y. Chang, H. Chen, and X. Hu, "Llm maybe longlm: Self-extend llm context window without tuning," in *Proc. of ICML*, 2024.

[75] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *Proc. of ICLR*, 2024.

[76] A. Chavan, R. Magazine, S. Kushwaha, M. Debbah, and D. Gupta, "Faster and lighter llms: A survey on current challenges and way forward," in *Proc. of IJCAI*, 2024.