Large language models for game development: A mapping study on automated code generation

Alireza Dastmalchi Saei[1], Mohammadreza Sharbaf[2] and Shekoufeh Kolahdouz-Rahimi[3,*]

[1]Department of Computer Engineering, Bilkent University, Ankara, Turkey
[2]Department of Software Engineering, University of Isfahan, Isfahan, Iran
[3]School of Arts, University of Roehampton, London, UK

Abstract

Large Language Models (LLMs) are increasingly utilised in software development, particularly for automatic code generation. Game development, a complex and multidisciplinary field, has begun leveraging LLMs to streamline various stages of the development process. This paper focuses on the application of LLMs in automating code generation for game development, providing a comprehensive survey of existing research from multiple perspectives. We analyse the benefits and limitations of LLM-based approaches in this domain and identify key challenges. Our systematic analysis of 15 primary studies reveals a prevalence of general-purpose LLMs (47%) alongside fine-tuned models (40%), with prompt engineering universally adopted (100%) to guide code generation. Our findings highlight the potential of LLMs to enhance game development workflows, while outlining future research directions to address existing limitations.

Keywords

Game Development, Code Generation, Large Language Models (LLMs), Systematic Review

1. Introduction

Large Language Models (LLMs) represent a paradigm shift in artificial intelligence, leveraging vast datasets to achieve human-like language understanding and generation capabilities [1, 2]. These models have demonstrated transformative potential across diverse domains, from machine translation to content creation and conversational agents. More recently, their application has extended to game development, where they promise to revolutionize one of the most labor-intensive aspects of the process: automated code generation.

Game development is a complex, multidisciplinary endeavor that requires seamless integration of design, scripting, and testing [3]. Traditionally, coding game mechanics—ranging from Non-Player Character (NPC) behavior to procedural content generation—has demanded significant time and expertise. LLMs, equipped with advanced natural language processing (NLP) capabilities, present a transformative approach to software development by enabling automated code generation, accelerating development timelines, and lowering the barriers to entry for independent developers and smaller development teams.. State-of-the-art models like GPT-4 [1], Code Llama [4], and their domain-specific variants have already shown promise in generating functional game code snippets, debugging scripts, and even optimizing performance-critical sections.

However, the integration of LLMs into game development is not without challenges. Domain-specific complexities, such as real-time rendering constraints, physics simulations, and platform-specific APIs, often lead to hallucinations or logically inconsistent output [3]. For instance, recent studies have found that LLMs underperform on game-specific code generation compared to

Joint Proceedings of the STAF 2025 Workshops: OCL, OOPSLE, LLM4SE, ICMM, AgileMDE, AI4DPS, and TTC. Koblenz, Germany, June 10-13, 2025

*Corresponding author.

dastmalchi@bilkent.edu.tr (A. D. Saei); m.sharbaf@eng.ui.ac.ir (M. Sharbaf); shekoufeh.rahimi@roehampton.ac.uk (S. Kolahdouz-Rahimi)
0000-0001-5113-7689 (M. Sharbaf); 0000-0002-0566-5429 (S. Kolahdouz-Rahimi)

general-purpose programming tasks, primarily due to difficulties in leveraging domain knowledge such as game engine APIs. To address these limitations, researchers have explored fine-tuning techniques, structured prompt engineering, and human-in-the-loop validation. Despite these advances, gaps remain in evaluating the reliability, efficiency, and scalability of LLM-generated code in production-grade game development workflows.

This paper presents a systematic survey of LLM applications for automated code generation in game development. We examine which LLMs are most widely adopted and how they are adapted to game-specific tasks, analyze the techniques used to optimize their performance, and identify persistent challenges and future research directions. By analyzing primary studies from 2019 to 2025, we provide a comprehensive synthesis of current practices, highlighting trends such as the dominance of general-purpose LLMs and the critical role of context-aware prompting. Our findings reveal that while LLMs excel at generating boilerplate code or scripting simple behaviors, challenges like controllability and the lack of standardized evaluation benchmarks hinder broader adoption.

The contributions of this work include a taxonomy of LLMs used in game development, a systematic analysis of optimization techniques and evaluation metrics, and actionable insights for practitioners and researchers. These insights emphasize the importance of hybrid human-AI workflows and domain-specific benchmarking to bridge the gap between experimental research and practical implementation.

The rest of this paper is organized as follows. We first review related work in the field in Section 2, then detail our research methodology in Section 3. Next, we present the results of our systematic analysis in Section 4, followed by a discussion of their implications in Section 5. Finally, we conclude with a summary of key findings and directions for future research in Section 6.

2. Related Work

The intersection of LLMs and digital games has recently garnered significant attention that led to initial efforts to survey this rapidly evolving landscape [3]. Understanding prior survey work and related research helps situate the specific contributions of our paper, which mainly focuses on the application of LLMs for automated code generation within the game development process based on a systematic analysis of existing primary studies.

Initial surveys have begun mapping the broader range. Yang et al. [5] provided a scoping review focused specifically on the GPT model series, they identify five key application areas: procedural content generation (PCG), mixed-initiative design, mixed-initiative gameplay, playing games, and game user research, based on a systematic keyword search of 55 articles published before 2024. Their work offers a valuable overview of GPT's diverse roles but is limited by its focus on a single LLM family and covers many applications beyond code generation. Gallotta et al. [6] presented a survey and roadmap that holds a wider, role-based typology derived from AI and games conference proceedings. They categorize LLM involvement as Player, Non-Player Character, Designer, Analyst, etc. and discuss potential and limitations within each role. Their 'Designer' role implicitly includes code generation aspects but the survey lacks a dedicated analysis of automated code generation as a specific task across different game development contexts. Pilaniwala et al. [7] proposed a futuristic workflow using Generative AI for democratizing casual game creation from prompts focused on architecture and high-level challenges rather than analyzing current code generation practices found in existing research.

More closely related is primary research investigating specific aspects of LLMs for code generation in relevant domains. Gu et al. [8] conducted an empirical study on LLM effectiveness (including ChatGPT, CodeLlama) for domain-specific code generation, explicitly including game development (C++) using libraries like Unreal Engine, Cocos2d-x, and Bgfx as one of their target domains. They found that LLMs underperform on domain-specific tasks compared to general code generation, attributing this to difficulties in utilizing domain-specific APIs. Consequently, their work focuses significantly on prompting strategies to inject domain knowledge and proposes a method called DomCoder, evaluating it with metrics like BLEU and CodeBLEU.

Our survey distinguishes itself from these prior works through its dedicated and systematic focus on automated code generation across various LLMs and specific GD tasks (e.g., testing, agent behavior scripting, PCG support code) as reported in the existing research literature (G01-G15). We provide a quantitative and qualitative analysis of the LLMs employed, the distinct tuning and prompt engineering techniques utilized for code generation, the evaluation metrics applied to assess the generated code artifacts, and the specific challenges and future directions highlighted by researchers actively working in this sub-domain. By concentrating on this specific software engineering application within game development, our survey offers targeted insights into current practices, limitations, and opportunities at the intersection of LLMs, Software Engineering, and Game Development.

3. Research Method

This study adopts a systematic mapping approach, following the well-established methodologies proposed by Brereton et al. [9] and Petersen et al. [10]. The research aims to address the following key questions:

- RQ1: Which LLMs have been employed in code generation for games?
- RQ2: What techniques are used to optimize and evaluate LLMs in game-related code generation?
- RQ3: What challenges and future directions exist for applying LLMs in game development code generation?

The study encompasses all relevant publications in the field from January 2020 to February 2025. The subsequent sections detail the two primary phases of the research protocol: (1) planning and (2) conducting. The findings are presented and discussed in Section 4.

3.1. Planning Phase

The planning phase of the research protocol involves defining the search strategy and review process. The search strategy is described here, while the review process is detailed in Section 3.2.

We organized search terms into three thematic groups: (A) LLMs, (B) Game Development, and (C) Code Generation. Terms within each group were connected using the Boolean OR operator, while requiring papers to match at least one term from all three groups (combined with AND logic) to ensure relevance. The specific keywords used for filtering studies are presented below.

- A = Large Language Model, LLM, Generative AI Model, Transformer, Pretrained Language Model, AI Language Model
- B = Game Development, Game Design, Game Engine, Games, Computer Games
- C = Code Generation, Automatic Coding, Automated Programming, Machine-Generated Code

These words can be combined to form the overall search string as follows:

Search String: A and B and C

Our systematic search began with Google Scholar, a comprehensive academic search engine, using an initial set of search terms. After compiling all potentially relevant papers, we removed duplicate entries. The first screening phase involved evaluating publications based on their titles, abstracts, keywords, introductions, and conclusions. Subsequently, we applied predefined inclusion and exclusion criteria to identify primary studies for further analysis while eliminating irrelevant publications.

Inclusion Criteria:

- Publications from 2020 to February 2025.
- Publications that use LLMs as the main assistant in the generation of code for game development.
- Publications in English.
- Publications from journals, conferences, and workshops.

Exclusion Criteria:

- Summary, survey, or review publications.
- Any publication other than journals, conferences, or workshops.
- Books, websites, technical reports, dissertations, tutorials.
- Publications containing models that cannot be executed or converted into executable game code.
- Publications that lack game domain features based on the NESI feature model [11].

To ensure literature coverage completeness, we employed backward snowballing by reviewing reference lists of key papers, supplemented with forward snowballing through citation tracking. All candidate papers underwent rigorous full-text evaluation against our inclusion criteria. When encountering multiple publications discussing variants of the same core method, we performed cross-source synthesis to develop comprehensive method profiles within our analysis framework.

3.2. Conducting Phase

In the second phase of our systematic review, we conduct the selection process using the search protocol we defined in the planning phase. The process of finding and choosing primary studies for our review is described in more detail in the following. We then present the data extraction process and report the systematic map extracted from the primary studies.

3.2.1. Study Selection

Figure 1 illustrates our multi-stage study selection process. Beginning with an initial database search and duplicate removal, we apply systematic filtering criteria before conducting snowballing and full-text analysis of candidate papers. The final phase consolidates our primary study set while incorporating relevant distributed findings from related publications.

We initiated the selection process by executing our predefined search queries in Google Scholar, which returned 103 potentially relevant articles. After removing duplicates, 91 studies advanced to the filtering stage. The first author created this initial list. The second author then reviewed and helped finalize this list. Subsequently, the first and second authors independently applied the iterative filtering steps (title/keyword/abstract screening, relevance check based on introduction/abstract, and formal inclusion/exclusion criteria), with disagreements resolved through discussion. The third author validated the outcomes of this filtering process. Through this process, we rejected 68 studies, retaining 23 for further analysis. The primary reasons for exclusion at this stage included a lack of focus on automated code generation.

To ensure comprehensive coverage, we conducted backward snowballing by examining references from these 23 studies, which identified 9 additional qualifying papers. This brought our candidate

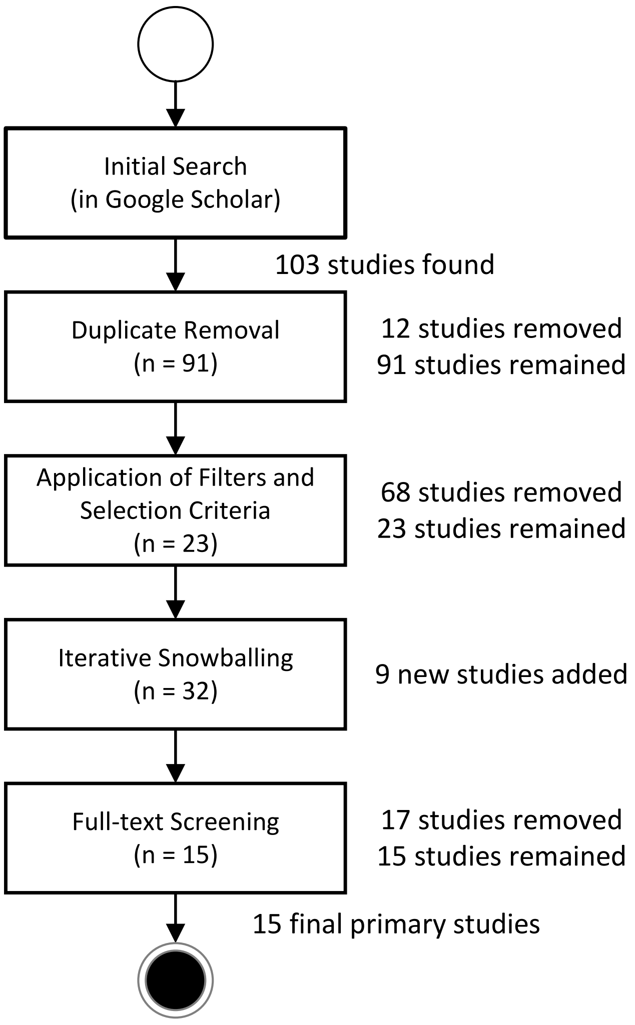

Figure 1: Primary studies selection process

pool to 32 studies. Subsequent full-text analysis excluded 17 papers that failed to meet our research objectives, resulting in 15 final primary studies for inclusion.

Figure 2 illustrates the annual publication trends of primary studies. The data reveal that employing LLMs for code generation in game development emerged as a novel research trend beginning in 2023, with a marked increase observable in 2024. Section 5 examines the potential factors driving this phenomenon.

3.2.2. Data Extraction and Synthesis

To address each research question, we manually extracted data from all primary studies using the standardized form presented in Table 1. This form captures essential study characteristics needed to answer our research questions. The first author completed all forms, which were then independently reviewed by the second and third authors. Any discrepancies identified during this review process were resolved through consensus.

4. Results

This section presents the findings of our systematic literature review based on the analysis of 15 primary studies (identified as G01-G15 and listed in Appendix A), structured to directly answer

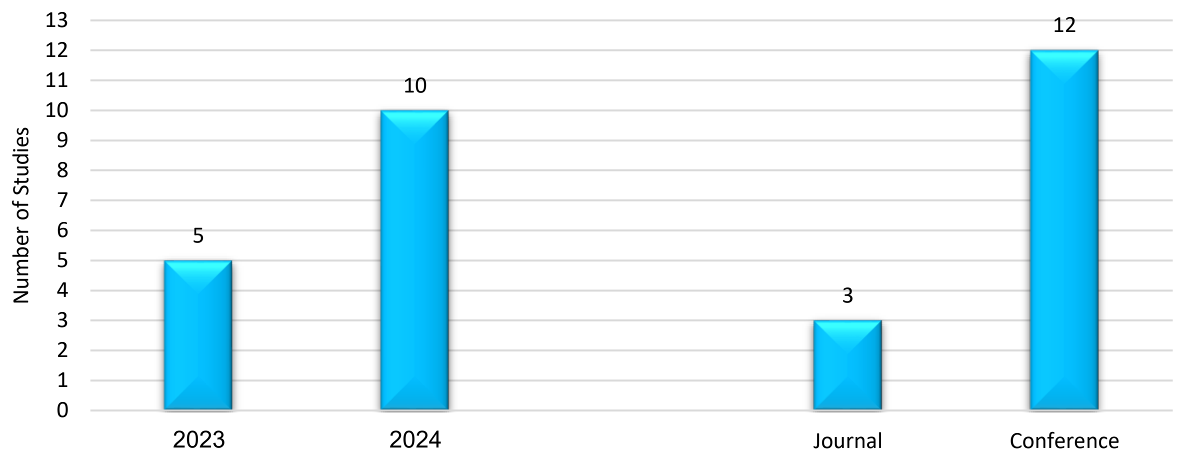

Figure 2: Primary studies publication trends over years

Table 1
Data Extraction Form

Study data	Description	Relevant RQ
Title		Study Overview
Author		Study Overview
Year		Study Overview
LLM Categories	What is the distribution of LLMs used in studies?	RQ1
Tuning Techniques	What are the applied tuning techniques?	RQ2
Prompt Engineering Techniques	What are the used prompt engineering techniques?	RQ2
Evaluation Metrics	What are the evaluation criteria in the study?	RQ2
Challenges	What are the challenges of the study?	RQ3
Future Directions	What are the suggested future directions?	RQ3

the research questions outlined in the Research Methodology Section. We report quantitative data derived from this analysis, supported by figures and tables.

4.1. RQ1: Which large language models (LLMs) have been employed in code generation for games?

Our analysis identified the LLMs used for automated code generation in the surveyed game development literature (N=15). General-purpose foundational models were the most frequent category, with models like OpenAI's GPT series, Google's Gemma, or Llama 2 identified as primary models or part of the toolchain in 7 papers (47%) [G02, G04, G05, G06, G09, G11, G15]. Access via ChatGPT or similar interactive tools was explicitly mentioned or implied in 6 studies [G03, G04, G11, G12, G13, G14].

Code-specific models, especially Meta's Code Llama, were used in 2 studies (13%) [G01, G07]. Importantly, fine-tuning was a key strategy, reported in 6 papers (40%), adapting various models (T5, GPT-2, GPT-3, RoBERTa, Code Llama) to game-specific contexts [G01, G02, G04, G08, G09, G10]. Integration with models for other modalities (image, voice) occurred in 2 papers (13%) [G03, G06]. Figure 3 visualizes the distribution of these LLM categories.

4.2. RQ2: What techniques are used to optimize and evaluate LLMs in game-related code generation?

This subsection gathers the techniques used to enhance and assess LLM performance for game development code generation tasks.

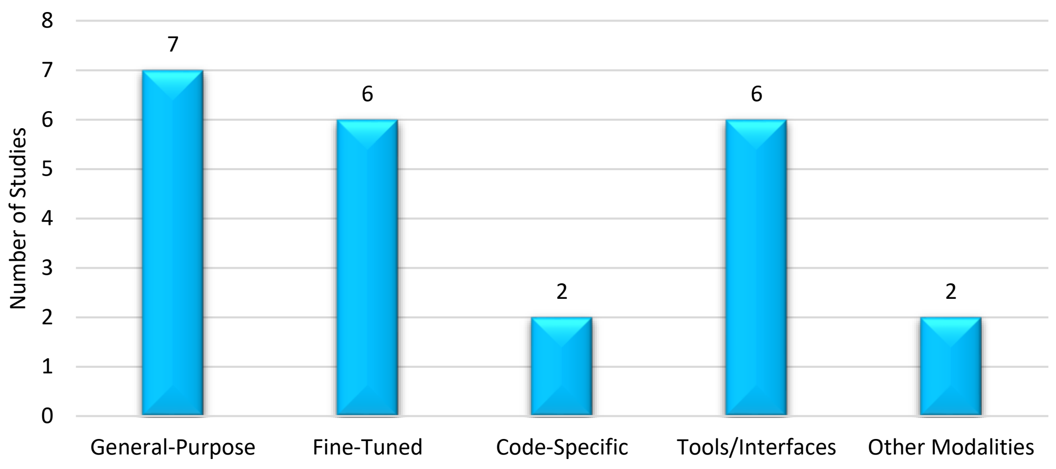

Figure 3: Distribution of LLM categories employed or referenced in the surveyed studies (Studies may utilize multiple categories)

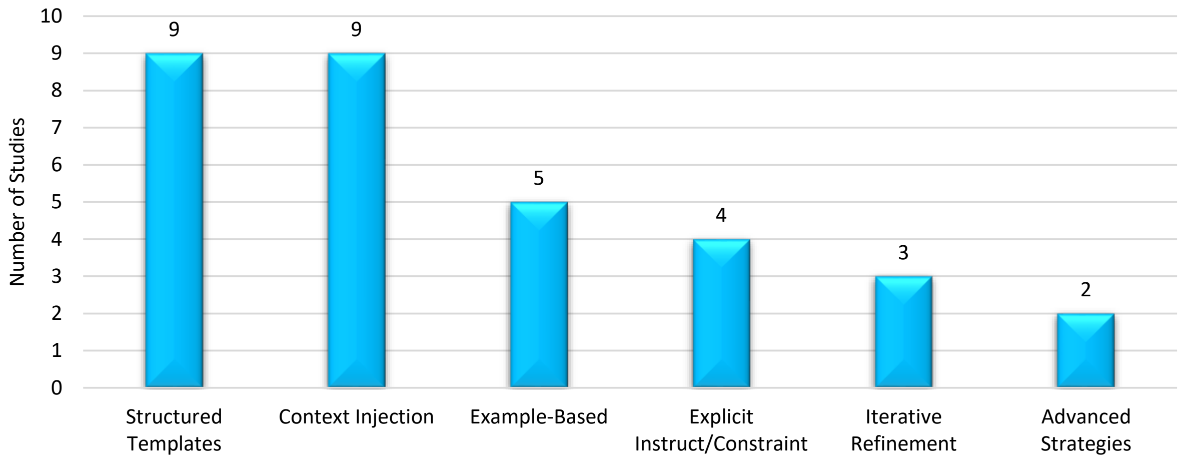

Figure 4: Frequency of reported prompt engineering techniques

4.2.1. RQ2.1: Tuning Techniques

Specific tuning techniques beyond prompting were described in several studies. Fine-tuning was the most common, reported in 6 studies (40%) [G01, G02, G04, G08, G09, G10]. This included using instruction datasets [G01] and Parameter-Efficient Fine-Tuning (PEFT) methods like LoRA [G01]. Data augmentation (2 papers, 13%) [G09, G10] and human-in-the-loop/bootstrapping (1 paper, 7%) [G10] were also employed.

4.2.2. RQ2.2: Prompt Engineering Techniques

Prompt engineering was applied across all 15 studies (100%), and structured templates or formatting were described in 9 papers (60%) [G01, G04, G06, G07, G10, G11, G13, G14, G15]. Context injection was also detailed in 9 papers (60%) [G01, G02, G05, G06, G08, G09, G11, G14, G15]. Additionally, example-based prompting was mentioned in 5 papers (33%) [G02, G05, G07, G12, G15]. Explicit instructions or constraints were highlighted in 4 papers (27%) [G10, G13, G14, G15]. Iterative refinement was discussed in 3 papers (20%) [G03, G04, G06]. Finally, advanced strategies (CoT, Function Calling) were explored in 2 papers (13%) [G11, G12]. Figure 4 illustrates the frequency of these techniques.

4.2.3. RQ2.3: Evaluation Metrics

A variety of metrics were employed. Code functionality/testing metrics were used in 1 study (7%) [G01], while task success/accuracy metrics were reported in 6 studies (40%) [G02, G07, G08, G09, G10, G14]. Metrics evaluating generated content quality/characteristics were most common in 7 studies (47%) [G05, G08, G09, G10, G12, G13, G15]. Additionally, Subjective/qualitative assessments were incorporated in 4 studies (27%) [G03, G06, G07, G14]. Finally, efficiency/robustness metrics were considered in 4 studies (27%) [G06, G11, G12, G13] and domain-specific metrics were used in 2 studies (13%) [G04, G14].

Table 2 summarizes the metric categories.

Table 2
Distribution of Evaluation Metric Categories Used (N=15). Studies may use multiple categories.

Metric Category	Number	Percentage (%)
Generated Content Quality / Characteristics	7	46.7
Task Success / Accuracy	6	40.0
Subjective / Qualitative Assessment	4	26.7
Efficiency / Robustness	4	26.7
Domain-Specific (Emotion, Narrative, etc.)	2	13.3
Code Functionality / Testing	1	6.7

5. Discussion

This section interprets the quantitative results from Section 4, synthesizes findings, discusses challenges and future directions (RQ3), outlines implications, and acknowledges limitations.

5.1. Synthesis of Findings

Our review (N=15) confirms LLMs are increasingly used for game code generation. The near-equal prevalence of general-purpose models (47%) and fine-tuned models (40%) (Figure 3) indicates a pragmatic balance between leveraging accessible, broad models and customizing models for domain specificity. The universal use of prompt engineering (100%), especially structured templates (60%) and context injection (60%) (Figure 4), highlights its necessity for guiding LLMs in this complex domain. The evaluation focus (Table 2) on content quality (47%) and task success (40%) suggests research often prioritizes the functional outcomes of generated code over intrinsic code metrics, apart from specific QA applications [G01]. The reviewed studies demonstrated LLM applications across various facets of game code generation, including the creation of unit tests [G01], scripting of agent behaviors [G07], and supporting procedural content generation (PCG) systems [G08, G09, G10].

5.2. RQ3: What challenges and future directions exist for applying LLMs in game development code generation?

5.2.1. RQ3.1: Challenges

Reliability and correctness remain the dominant challenge (11 studies, 73%), involving hallucinations and logical errors crucial for code [G03-G12, G14-G15]. Controllability and constraint adherence (8 studies, 53%) require significant effort via prompting and tuning [G02, G04, G07, G09, G11, G13, G14, G15]. Representing domain complexity (5 studies, 33%) [G01, G08-G10, G13], data scarcity (3 studies, 20%) [G01, G09, G10], and performance/efficiency (3 studies, 20%) [G02, G06, G07] are other key barriers. Ethical/human factors were also noted [G03].

5.2.2. RQ3.2: Future Directions

Future work frequently aims to improve techniques (prompting, tuning; 10 studies, 67%) and expand scope/functionality (9 studies, 60%) to address limitations and tackle more complex tasks [G05-G07, G09-G10, G12, G14-G15]. Directly addressing core limitations (5 studies, 33%) [G02, G04, G06, G07, G10], enhancing human-AI collaboration (4 studies, 27%) [G01, G03, G11, G15], improving evaluation/benchmarking (5 studies, 33%) [G02, G11-G14], and leveraging newer models (5 studies, 33%) [G02, G05, G09, G11, G14] are critical for advancement.

5.3. Implications

LLMs offer significant potential for accelerating game development, automating tasks like test generation [G01] or agent behavior scripting [G07]. This could lower barriers for complex features. However, the prevalence of reliability challenges (73% of studies) necessitates rigorous human oversight and validation. For researchers, this domain demands novel techniques addressing game-specific code complexities and robust verification methods. For game development specifically, this implies a need for LLMs that can understand and generate code compatible with complex game engine APIs (e.g., Unity, Unreal Engine), physics systems, and real-time rendering loops. The challenge lies not just in syntactic correctness but in semantic and performance appropriateness within the game's context.

5.4. Threats to Validity

This systematic review, while rigorously conducted on 15 primary studies, faces limitations in database selection, categorization, and keeping pace with rapid advancements. The primary validity threat stems from potential study selection biases, despite following Brereton et al. [9] and Petersen et al.'s guidelines [10]. Title-based filtering may have excluded relevant studies using alternate terminology, and while snowballing helped recover some omissions, restrictive criteria or incomplete search terms may still have missed pertinent research in this fast-evolving field.

6. Conclusion

The integration of LLMs into game development for automated code generation represents a promising frontier with significant potential to revolutionize the industry. Our systematic review of 15 primary studies highlights the growing adoption of general-purpose and code-specific LLMs, alongside the universal reliance on prompt engineering and fine-tuning techniques to address domain-specific challenges. While LLMs demonstrate remarkable capabilities in generating functional code, issues such as reliability, controllability, and domain complexity remain critical barriers to widespread adoption.

Future research should focus on improving model reliability, expanding the scope of LLM applications, and developing robust evaluation frameworks tailored to game development. Additionally, fostering human-AI collaboration and leveraging advancements in model architectures will be key to unlocking the full potential of LLMs in this domain. As the field continues to evolve, this survey serves as a foundational resource for researchers and practitioners, offering insights into current trends, challenges, and opportunities at the intersection of LLMs and game development. By addressing these challenges, the community can pave the way for more efficient, accessible, and innovative game development workflows.

Declaration on Generative AI

During the preparation of this work, the authors used generative AI tools (ChatGPT) to support grammar and spelling checking. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publications content.

References

[1] W. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, A survey of large language models, arXiv 2303.18223v10 (2023).

[2] A. Hemmat, M. Sharbaf, S. Kolahdouz-Rahimi, K. Lano, S. Y. Tehrani, Research directions for using llm in software requirement engineering: a systematic review, Frontiers in Computer Science 7 (2025) 1519437.

[3] A. Payandeh, M. Sharbaf, S. K. Rahimi, A systematic review of model-driven game development studies, IEEE Transactions on Games (2024).

[4] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, et al., Code llama: Open foundation models for code, arXiv preprint arXiv:2308.12950 (2023).

[5] D. Yang, E. Kleinman, C. Harteveld, Gpt for games: A scoping review (2020-2023), in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–8.

[6] R. Gallotta, G. Todd, M. Zammit, S. Earle, A. Liapis, J. Togelius, G. N. Yannakakis, Large language models and games: A survey and roadmap, IEEE Transactions on Games (2024).

[7] P. Pilaniwala, G. Chhabra, P. Kaur, The future of game development in the era of gen ai, in: 2024 Artificial Intelligence for Business (AIxB), IEEE, 2024, pp. 39–42.

[8] X. Gu, M. Chen, Y. Lin, Y. Hu, H. Zhang, C. Wan, Z. Wei, Y. Xu, J. Wang, On the effectiveness of large language models in domain-specific code generation, ACM Transactions on Software Engineering and Methodology 34 (2025) 1–22.

[9] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software 80 (2007) 571–583.

[10] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Information and Software Technology 64 (2015) 1–18.

[11] V. Sarinho, A. Apolinário, A feature model proposal for computer games design, in: VII Brazilian Symposium on Computer games and Digital entertainment, Belo horizonte, 2008, pp. 54–63.

[12] C. Paduraru, A. Staicu, A. Stefanescu, Llm-based methods for the creation of unit tests in game development, Procedia Computer Science 246 (2024) 2459–2468.

[13] A. Goslen, Y. J. Kim, J. Rowe, J. Lester, Llm-based student plan generation for adaptive scaffolding in game-based learning environments, International journal of artificial intelligence in education (2024) 1–26.

[14] J. D. Boucher, G. Smith, Y. D. Telliel, Is resistance futile?: Early career game developers, generative ai, and ethical skepticism, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–13.

[15] A. Marincioni, M. Miltiadous, K. Zacharia, R. Heemskerk, G. Doukeris, M. Preuss, G. Barbero, The effect of llm-based npc emotional states on player emotions: An analysis of interactive game play, in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–6.

[16] C. Hu, Y. Zhao, J. Liu, Game generation via large language models, in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–4.

[17] P. Mieschke, S. Radicke, Adaptive llm-based game radio (algr), in: 2023 4th International

Conference on Computers and Artificial Intelligence Technology (CAIT), IEEE, 2023, pp. 277–283.

[18] R. Ito, J. Takahashi, Game agent driven by free-form text command: Using llm-based code generation and behavior branch (in press), in: The 38th Annu. Conf. of the, 2024.

[19] S. Sudhakaran, M. González-Duque, M. Freiberger, C. Glanois, E. Najarro, S. Risi, Mariogpt: Open-ended text2level generation through large language models, Advances in Neural Information Processing Systems 36 (2023) 54213–54227.

[20] G. Todd, S. Earle, M. U. Nasir, M. C. Green, J. Togelius, Level generation through large language models, in: Proceedings of the 18th International Conference on the Foundations of Digital Games, 2023, pp. 1–8.

[21] M. U. Nasir, J. Togelius, Practical pcg through large language models, in: 2023 IEEE Conference on Games (CoG), IEEE, 2023, pp. 1–4.

[22] R. Gallotta, A. Liapis, G. Yannakakis, Consistent game content creation via function calling for large language models, in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–4.

[23] P. Taveekitworachai, F. Abdullah, M. F. Dewantoro, Y. Xia, P. Suntichaikul, R. Thawonmas, J. Togelius, J. Renz, Chatgpt4pcg 2 competition: Prompt engineering for science birds level generation, in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–8.

[24] P. Taveekitworachai, F. Abdullah, M. F. Dewantoro, R. Thawonmas, J. Togelius, J. Renz, Chatgpt4pcg competition: character-like level generation for science birds, in: 2023 IEEE Conference on Games (CoG), IEEE, 2023, pp. 1–8.

[25] T. Rist, Using a large language model to turn explorations of virtual 3d-worlds into interactive narrative experiences, in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–8.

[26] S. Hu, Z. Huang, C. Hu, J. Liu, 3d building generation in minecraft via large language models, in: 2024 IEEE Conference on Games (CoG), IEEE, 2024, pp. 1–4.

A. List of Primary Studies Reviewed

Table 3 lists the primary research papers included in the quantitative analysis of this survey, referenced by their G# identifier.

Table 3: Primary Studies Included in the Survey Analysis (N=15)

Ref. ID	Title of Paper
G01 [12]	LLM-based methods for the creation of unit tests in game development
G02 [13]	LLM-Based Student Plan Generation for Adaptive Scaffolding in Game-Based Learning Environments
G03 [14]	Is Resistance Futile? Early Career Game Developers, Generative AI, and Ethical Skepticism
G04 [15]	The Effect of LLM-Based NPC Emotional States on Player Emotions: An Analysis of Interactive Game Play
G05 [16]	Game Generation via Large Language Models
G06 [17]	Adaptive LLM-based Game Radio (ALGR)
G07 [18]	Game Agent Driven by Free-Form Text Command: Using LLM-based Code Generation and Behavior Branch
G08 [19]	Open-Ended Text2Level Generation through Large Language Models
G09 [20]	Level Generation Through Large Language Models
G10 [21]	Practical PCG Through Large Language Models

Continued on next page

Table 3: Primary Studies Included in the Survey Analysis (N=15) – continued

Ref. ID	Title of Paper
G11 [22]	Consistent Game Content Creation via Function Calling for Large Language Models
G12 [23]	ChatGPT4PCG 2 Competition: Prompt Engineering for Science Birds Level Generation
G13 [24]	ChatGPT4PCG Competition: Character-like Level Generation for Science Birds
G14 [25]	Using a Large Language Model to turn Explorations of Virtual 3D-Worlds into Interactive Narrative Experiences
G15 [26]	3D Building Generation in Minecraft via Large Language Models