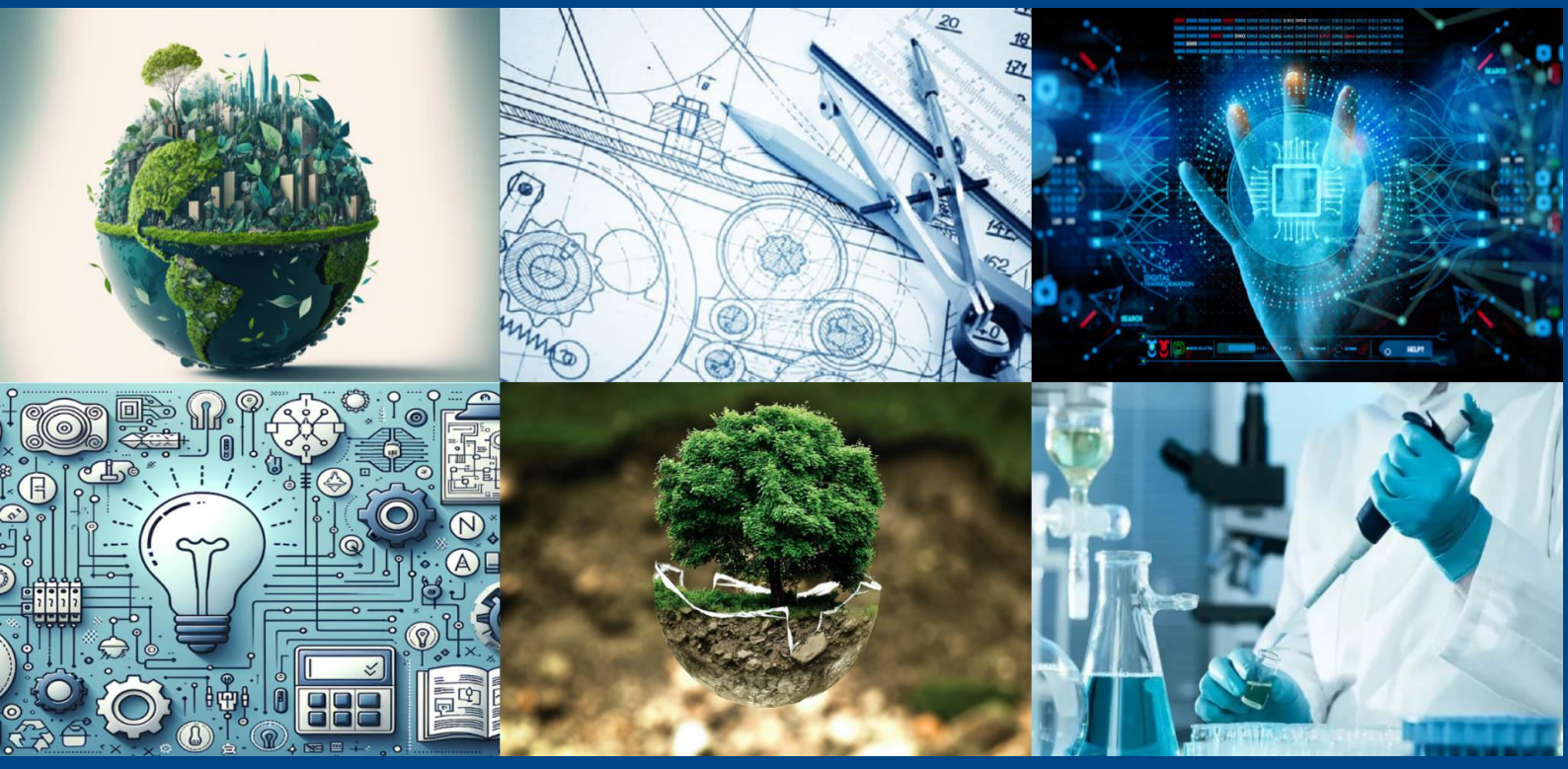
ISSN: 2582-7219

International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.206 **Volume 8, Issue 5, May 2025**

© 2025 IJMRSET | Volume 8, Issue 5, May 2025| DOI:10.15680/IJMRSET.2025.0805113

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Study of Performance Improvement Techniques for Code Generation in Large Language Models

Apeksha M. Sonawane, Prof. Purvesh Wagh

PG Student, Dept. of MCA, Anantrao Pawar College of Engineering & Research, Pune, India

Assistant Professor, Dept. of MCA, Anantrao Pawar College of Engineering & Research, Pune, India

ABSTRACT: Recent advancements in Artificial Intelligence (AI) and Machine Learning (ML) have led to the emergence of Large Language Models (LLMs) capable of generating human-like text and code. Despite their capabilities, generating accurate and efficient code remains a significant challenge. This study explores techniques aimed at enhancing LLM performance in code generation, focusing on Fine-Tuning, Prompt Design, and Context Awareness.

Fine-Tuning involves adapting LLMs to specific coding tasks using targeted datasets, improving their task-specific performance. Prompt Design emphasizes crafting effective input prompts to guide LLMs toward more relevant and accurate code outputs. Context Awareness ensures the model maintains coherence and utilizes in-file and cross-file information effectively.

Through empirical evaluations and case studies, this survey assesses the impact of these techniques on the quality and efficiency of code generation. The findings provide insights and guidelines to optimize LLMs, contributing to more robust code generation and advancing software development practices.

KEYWORDS: Large Language Models (LLMs), Fine-Tuning, Prompt Design, Context Awareness, Code Generation

I. INTRODUCTION

The proliferation of LLMs has opened new frontiers in automating software development through code generation. However, challenges remain in achieving reliable, accurate, and contextually relevant code. This paper reviews key performance enhancement strategies that improve the code generation capabilities of LLMs.
The study focuses on three major techniques:

• Fine-Tuning, which customizes models for specific domains using curated datasets.
• Prompt Design, which influences output quality by structuring input queries effectively.
• Context Awareness, which allows models to leverage broader contextual information to maintain consistency across code files and functions. By analyzing these methods, the paper offers a comprehensive overview of current advancements and their effectiveness in enhancing LLM-driven code generation.

II. FINE-TUNING IN LARGE LANGUAGE

Fine-Tuning is the process of refining a pre-trained LLM using task-specific datasets, enabling it to specialize in targeted applications like code generation. This process involves adjusting model parameters via gradient descent to improve performance while preserving general language comprehension. Fine-tuning enhances contextual understanding and domain relevance by leveraging smaller, focused datasets. This is often achieved using a minimal set of human-labeled examples that serve as seed data for generating larger training corpora. The goal is to maximize relevance while minimizing manual effort. Traditional approaches include:

• Supervised Fine-Tuning, which uses labeled datasets.
• Transfer Learning, where models are adapted from general tasks to domain-specific ones.
• Domain Adaptation, tailored to specific programming environments or languages. Recent advancements also explore parameter-efficient fine-tuning techniques, such as adapter layers or low-rank adaptation (LoRA), which reduce computational costs while retaining performance benefits.

© 2025 IJMRSET | Volume 8, Issue 5, May 2025| DOI:10.15680/IJMRSET.2025.0805113

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

III. PROMPT DESIGN IN CODE GENERATION

• Prompt design is an essential strategy in maximizing the accuracy and relevance of code generated by large language models (LLMs). At its core, it's about thoughtfully shaping the input given to the model to produce responses that align with the intended outcome especially in technical domains like programming.
• Effective prompt engineering hinges on a few key principles:
• Clarity and Specificity: Vague or loosely defined prompts often result in generic or off-target code. Clearly defined instructions help steer the model toward the desired solution.
• Contextual Cues: Supplying examples—such as code snippets, explanatory comments, or expected outputs—enables the model to better grasp the format and behavior you're aiming for.
• Instructional Framing: Tailoring prompts to include explicit directions, like the programming language, constraints, or formatting preferences, can significantly refine the model's responses.
• Prompt design becomes especially impactful when leveraged alongside few-shot or zero-shot learning. In these modes, the model generates useful results from minimal (or even no) prior examples, inferring structure and logic on the fly.
• Moreover, the layout and flow of information in a prompt can shape the quality of output. Positioning critical details early and maintaining a consistent format tends to boost response coherence. Given the high sensitivity of LLMs to prompt structure, even minor tweaks can lead to dramatically different results—making prompt crafting not just a skill, but a form of art in AI-driven development.
• language, logic constraints, formatting styles, or edge cases—can dramatically enhance the quality and consistency of the results.

IV. CONTEXT AWARENESS IN CODE GENERATION

LLMs traditionally generate code in a stateless manner, which can lead to issues with consistency and logic, especially in large or modular projects. Context Awareness addresses this by enabling models to understand and remember relevant information from surrounding code.

There are two primary types of context:
• In-File Context: Maintaining coherence within a single file, including function calls, variable declarations, and comments.
• Cross-File Context: Understanding dependencies and interactions across multiple files or modules in a codebase.
To improve context awareness, techniques such as:
• Context Window Expansion: Extending the token window size to capture more surrounding code.
• Chunking and Memory Techniques: Dividing code into logical chunks and feeding them sequentially to maintain continuity.
• Embedding-Based Retrieval: Using vector similarity to fetch and present relevant code blocks as context before generation. Context-aware models reduce the chances of redundant or conflicting code and enhance the model's ability to produce syntactically and semantically valid outputs across complex software projects.

V. EMPIRICAL ANALYSIS AND CASE STUDIES

To evaluate the effectiveness of the three primary techniques—Fine-Tuning, Prompt Design, and Context Awareness—this study synthesizes data and findings from recent benchmarks, research papers, and tool performance reports such as HumanEval, CodeXGLUE, and MBPP.

5.1 Evaluation Matrix Code generation performance is measured using the following metrics:

• Exact Match Accuracy (EMA): Measures if the generated code exactly matches the reference code.
• Pass@k: Assesses if the correct code is generated within k attempts.
• Execution Accuracy: Validates whether the code produces the correct output.
• BLEU/CodeBLEU Score: Evaluates similarity in syntax and semantics.
• Human Evaluation: Assesses code readability, maintainability, and logic.

© 2025 IJMRSET | Volume 8, Issue 5, May 2025| DOI:10.15680/IJMRSET.2025.0805113

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

5.2 Comparative Performance Overview

Technique	Improvement Observed Notable Tools/Models Used.
Fine-Tuning Prompt	+25-30% EMA CodeT5, StarCoder, Codex
Prompt Design	+10-20% CodeBLEU GitHub Copilot, ChatGPT
Context Awareness	+15-25% Execution Acc CodeBERT, PolyCoder

5.3 Case Study Example

A comparative test on a code generation task (e.g., creating a binary search algorithm) revealed:
• A fine-tuned model provided concise, optimized code with fewer logic errors.
• A well-structured prompt led to more readable and properly commented output.
• Including relevant contextual code (e.g., previous function definitions) resulted in correct variable usage and consistent naming conventions.

VI. DISCUSSION AND RECOMMENDATIONS

Each technique offers unique strengths but also comes with trade-offs:

• Fine-Tuning is ideal for domain-specific tasks (e.g., legal, medical, or scientific software) but requires significant computational resources and curated datasets.
• Prompt Design is highly flexible and low-cost but demands expertise in designing efficient prompts, and results can vary with slight changes in input.
• Context Awareness boosts performance in large codebases and long files, but managing token limits and context overflow remains a technical challenge. Recommendations:
• Combine Fine-Tuning with Prompt Design for high-stakes or specialized code tasks.
• Leverage retrieval-based context augmentation to maintain coherence in large projects.
• Use parameter-efficient fine-tuning (e.g., adapters, LoRA) for resourcelimited environments.
• Automate prompt generation for consistent results across use cases.

VII. CONCLUSION

The study highlights the crucial role of Fine-Tuning, Prompt Design, and Context Awareness in enhancing LLM-based code generation. Each technique addresses a specific aspect of model performance—from domain alignment and input guidance to maintaining logical flow and consistency. As LLMs continue to evolve, integrating these techniques can lead to more reliable, efficient, and contextually intelligent code generation systems. Future research should explore hybrid strategies, automated toolchains for prompt/context management, and more nuanced evaluation methods to further elevate the role of LLMs in software development.

REFERENCES

1) Jia Li, Yunfei Zhao, Yongmin Li, Ge Li, Zhi Jin, “AceCoder: Utilizing Existing Code to Enhance Code Generation”, 2023.
2) Chao Liu , Xuanlin Bao , Hongyu Zhang , Neng Zhang , Haibo Hu , Xiaohong Zhang , Meng Yan, “Improving ChatGPT Prompt for Code Generation”, 2023.
3) Huaiyu Guo, “ An empirical study of prompt mode in code generation based on ChatGPT”, 2024.
4) Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128 (2023)
5) Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. 2023. Impact of code language models on automated program repair. arXiv preprint arXiv:2302.05020 (2023)
6) Yikun Wang et al. "Rescue: Ranking LLM Responses to Enhance Reasoning Over Context." ArXiv, abs/2311.09136 (2023).

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

7) Jia Li et al. "Large Language Model-Aware In-Context Learning for Code Generation." ArXiv, abs/2310.09748 (2023).
8) Hengzhi Pei et al. "Better Context Makes Better Code Language Models: A Case Study on Function Call Argument Completion." ArXiv, abs/2306.00381 (2023).
9) D.L. Parnas, P.C. Clements, and D.M. Weiss. 1985. The modular structure of complex systems. IEEE Transactions on Software Engineering, SE-11(3):259–266.
10) Li, Yichen, Yun Peng, Yintong Huo, and Michael R. Lyu. "Enhancing LLM-Based Coding Tools through Native Integration of IDE-Derived Static Context." arXiv:2402.03630 (2024)
11) Kaixin Li,Qisheng Hu,Xu Zhao ,Hui Chen,Yuxi Xie , Tiedong Liu,Qizhe Xie1,Junxian He "InstructCoder: Instruction Tuning Large Language Models for Code Editing" http://arxiv.org/abs/2310.20329 (2024).
12) MARTIN WEYSSOW, XIN ZHOU,KISUB KIM,DAVID LO,HOUARI SAHRAOUI "Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models" arXiv:2308.10462v2 [cs.SE] (2024).
13) Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. "Qlora: Efficient finetuning of quantized llms" arXiv preprint arXiv:2305.14314 (2023)
14) Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan,Weize Chen, et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained anguage models. arXiv preprint arXiv:2203.06904 (2022).

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com