Scaling LLMs: An Empirical Study on LLaMA-Based Architectures for Python Code Generation

Natural Language Processing (BCSE409L)

PROJECT REPORT

Fall Semester 2025-2026

Submitted by

Aattreya K.S 22BCE0078

Rushil Shamsher Sethi 22BCE0783

Harshil Gandhi 22BCE0705

in partial fulfillment for the award of the degree of

B. Tech

in

Computer Science and Engineering

Submitted to

Dr. Hiteshwar Kumar Azad

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vellore-632014, Tamil Nadu, India

School of Computer Science and Engineering

November, 2025

Table of Contents

Abstract
A concise summary outlining the purpose, scope, and contributions of the study.

I. Introduction
A. Background
Overview of large language models and their influence on software automation.
B. Problem Statement
Defining the research problem and scaling objectives for LLaMA-based architectures.
C. Objectives
Listing the study's key aims, including scaling analysis and performance evaluation.

II. Methodology
A. Model Architecture
Description of LLaMA-based design, embedding mechanisms, and configuration consistency.
B. Training Methodology
Details of the training process, datasets, optimization strategies, and resource setup.
C. Evaluation Benchmarks
Explanation of metrics, datasets (HumanEval, MBPP, APPS), and performance criteria.

III. Results and Discussion
Analysis of scaling outcomes, performance patterns, and emergent reasoning behaviors across model sizes.

IV. Limitations
Discussion of computational constraints, dataset scope, and evaluation boundaries impacting study generalization.

V. Future Work
Suggestions for extending the research through multi-language models, instruction tuning, and larger context analysis.

VI. Conclusion
Summary of key findings, highlighting scaling relationships and implications for efficient LLM-based code generation.

References
List of cited works following IEEE referencing conventions.

Abstract

This paper investigates the scaling behavior of LLaMA-based architectures in the context of Python code generation. Five models, ranging from 10 million to 7 billion parameters, are evaluated to examine how model scale influences code quality, efficiency, and emergent reasoning capabilities. The primary objective is to determine the minimal viable configuration that exhibits structured and functionally meaningful Python code generation. The study analyzes compute-performance trade-offs and emergent phenomena associated with model size, providing empirical insights into scaling laws for code-oriented large language models (LLMs). The findings aim to guide future architectural design and model selection for efficient and accurate code generation tasks.

Keywords— LLaMA, Large Language Models, Code Generation, Python, Scaling Laws, Transformer Architecture

I. INTRODUCTION

A. Background

Large Language Models (LLMs) have revolutionized software development by enabling automation in tasks such as code synthesis, completion, and debugging. Architectures such as LLaMA, Codex, and CodeLLaMA leverage transformer-based mechanisms that exhibit exceptional performance in both natural and programming language generation. However, while scaling laws have been extensively studied in natural language processing, their empirical characterization in structured programming languages—especially Python—remains relatively unexplored.

Scaling fundamentally impacts a model's internal representations and cognitive-like behaviors, influencing its ability to produce syntactically coherent and logically structured code. Understanding this relationship is crucial to achieving high-quality code generation without incurring unnecessary computational cost. This work aims to bridge this knowledge gap by systematically evaluating the effect of model scale on Python code generation using LLaMA-based architectures.

B. Problem Statement

This study addresses the question: *How does the scale of a LLaMA-based model affect the quality, structure, and efficiency of Python code generation?*

Five model variants were designed to evaluate this relationship:

Model ID	Description	Parameter Count
M1	Minimal	10M
M2	Small	100M
M3	Medium	500M
M4	Large	3B
M5	Full	7B

The research focuses on three key aspects:

1. How model scale influences code generation quality across varying complexity levels.
2. The trade-offs between computational cost and output performance.
3. The parameter threshold at which emergent coding behaviors such as logical flow and abstraction handling become evident.

C. Objectives

The objectives of this study are as follows:

1. Quantify the relationship between model size and code generation proficiency.
2. Characterize trade-offs between computational efficiency and generation performance.
3. Identify scaling patterns and emergent properties in LLaMA-based code generation.
4. Provide empirically grounded guidelines for model selection in practical applications.

II. METHODOLOGY

A. Model Architecture

All models in this study are based on the LLaMA architecture, ensuring architectural consistency across scales. The key components and design principles are summarized below:

- **Rotary Positional Embeddings (RoPE):** Enable stable positional encoding for long sequences.
- **RMSNorm:** Provides computationally efficient normalization and improved training stability.

- **SwiGLU Activation:** Enhances non-linearity and gradient flow.
- **Causal Attention Masking:** Ensures autoregressive sequence modeling.
- **Bias-Free Linear Layers:** Reduce parameter redundancy.
- **Vocabulary Size:** 32,000 tokens.
- **Context Length:** 2,048 tokens.
- **Rotary Theta:** 10,000.

By maintaining identical architectural configurations across all models, performance variations can be directly attributed to differences in scale rather than design discrepancies.

B. Training Methodology

Each model was pre-trained on the **BigCode/The Stack v2** dataset available via Hugging Face, consisting of a diverse and licensed corpus of Python source code. The dataset includes algorithmic, functional, and object-oriented code, ensuring comprehensive language coverage.

Training Configuration:

- **Tokenizer:** Byte-Pair Encoding (BPE) with 32,000-token vocabulary.
- **Optimizer:** AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay = 0.1.
- **Gradient Clipping:** 1.0 for stability.
- **Learning Rate Schedule:** Linear warmup followed by cosine decay.
- **Batch Size:** Proportional to model scale to balance compute usage.
- **Total Tokens Trained:** 32,000 (controlled subset).
- **Hardware:** Single GPU environment (to be specified).

A uniform training pipeline was adopted to eliminate confounding effects from hyperparameter tuning, ensuring that performance differences reflect the impact of scale alone.

C. Evaluation Benchmarks

Evaluation was conducted using three established benchmarks tailored for Python code generation:

1. **Perplexity:** Measures intrinsic model quality on a held-out Python corpus, reflecting language modeling efficiency.
2. **HumanEval:** A benchmark of 164 function-level tasks assessing correctness and reasoning.

3. **MBPP (Mostly Basic Programming Problems):** Comprising 974 simple tasks focused on syntactic accuracy and functional correctness.
4. **APPS (Introductory Subset):** Tests algorithmic reasoning and real-world code generation ability.

Metrics Used:

- **Perplexity (bits per byte):** Indicates the predictive capability of the model.
- **Pass@1 and Pass@10:** Reflect functional correctness and logical consistency.
- **Aggregate Accuracy:** Measures overall success across benchmarks.

The comparative analysis will identify scaling thresholds, performance saturation points, and emergent reasoning patterns as model size increases.

III. RESULTS AND DISCUSSION

(To be completed post-inference)

Preliminary expectations suggest a nonlinear scaling effect where significant performance improvements occur between 100M and 3B parameters, beyond which marginal gains diminish relative to compute cost. Models exceeding 500M parameters are hypothesized to begin demonstrating emergent structural coding behavior, including self-consistent syntax generation, abstraction, and modular reasoning. These observations will be validated empirically upon completion of inference and benchmarking.

IV. LIMITATIONS

This study is constrained by several factors:

- **Computational Resources:** Training is limited to single-GPU configurations, restricting larger-scale experiments.
- **Dataset Scope:** Focused solely on Python, limiting generalization to other programming languages.
- **Evaluation Method:** Automated metrics are used without human qualitative assessment.
- **Architectural Rigidity:** The base LLaMA architecture is fixed without architectural optimization or tuning.

Despite these constraints, the methodology provides a controlled framework for analyzing scaling trends in code generation tasks.

V. FUTURE WORK

Future research directions include:

1. **Multi-Language Expansion:** Training models on multilingual code datasets to assess generalization across languages.
2. **Instruction Tuning and Alignment:** Incorporating reinforcement learning from human feedback (RLHF) for more adaptive, context-aware generation.
3. **Mixture-of-Experts (MoE) Architectures:** Employing modular scalability for compute efficiency.
4. **Human-Centric Evaluation:** Including subjective assessments of readability, maintainability, and style.
5. **Extended Context Windows:** Enabling reasoning across larger codebases and multi-file projects.

These extensions will support a broader understanding of scaling behavior and enhance practical deployment potential.

VI. CONCLUSION

This paper presents a systematic empirical investigation into the scaling effects of LLaMA-based architectures for Python code generation. By analyzing models spanning 10M to 7B parameters, it aims to establish the relationship between scale, emergent behavior, and computational efficiency. The study emphasizes the discovery of the minimal configuration capable of generating syntactically valid and semantically meaningful Python code. The outcomes are expected to contribute to the design of optimized, resource-efficient LLMs tailored for programming tasks and to guide future research on scaling laws in code generation.

REFERENCES

[1] Touvron, H. *et al.*, “LLaMA: Open and Efficient Foundation Language Models,” Meta AI Research, 2023.
[2] Chen, M. *et al.*, “Evaluating Large Language Models Trained on Code,” *arXiv preprint arXiv:2107.03374*, 2021.
[3] Li, Y. *et al.*, “Competition-Level Code Generation with AlphaCode,” *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.
[4] BigCode Project, “The Stack v2 Dataset,” Hugging Face, 2023.
[5] Kaplan, J. *et al.*, “Scaling Laws for Neural Language Models,” *arXiv preprint arXiv:2001.08361*, 2020.