International Journal of Electrical and Computer Engineering (IJECE)
Vol. 16, No. 1, February 2026, pp. 278~287
ISSN: 2088-8708, DOI: 10.11591/ijece.v16i1.pp278-287

Parameter-efficient fine-tuning of small language models for code generation: a comparative study of Gemma, Qwen 2.5 and Llama 3.2

Van-Viet Nguyen[1], The-Vinh Nguyen[1], Huu-Khanh Nguyen[2], Duc-Quang Vu[1]
[1]Faculty of Information Technology, Thai Nguyen University of Information and Communication Technology, Thai Nguyen, Vietnam
[2]University of Information and Communications Technology, Thai Nguyen University, Thai Nguyen, Vietnam

Article Info

Article history:

Received May 11, 2025
Revised Oct 2, 2025
Accepted Nov 23, 2025

Keywords:

Fine-tuning
SLMCode
Small device
Small language models
Software engineering

ABSTRACT

Large language models (LLMs) have demonstrated impressive capabilities in code generation; however, their high computational demands, privacy limitations, and challenges in edge deployment restrict their practical use in domain-specific applications. This study explores the effectiveness of parameter efficient fine-tuning for small language models (SLMs) with fewer than 3 billion parameters. We adopt a hybrid approach that combines low-rank adaptation (LoRA) and 4-bit quantization (QLoRA) to reduce fine-tuning costs while preserving semantic consistency. Experiments on the CodeAlpaca-20k dataset reveal that SLMs fine-tuned with this method outperform larger baseline models, including Phi-3 Mini 4K base, in ROUGE-L. Notably, applying our approach to the LLaMA 3 3B and Qwen2.5 3B models yielded performance improvements of 54% and 55%, respectively, over untuned counterparts. We evaluate models developed by major artificial intelligence (AI) providers Google (Gemma 2B), Meta (LLaMA 3 1B/3B), and Alibaba (Qwen2.5 1.5B/3B) and show that parameter-efficient fine-tuning enables them to serve as cost-effective, high-performing alternatives to larger LLMs. These findings highlight the potential of SLMs as scalable solutions for domain-specific software engineering tasks, supporting broader adoption and democratization of neural code synthesis.

Corresponding Author:

Van-Viet Nguyen
Faculty of Information Technology, Thai Nguyen University of Information and Communication Technology
Z115 Road, Thai Nguyen 250000, Vietnam
Email: nvviet@ictu.edu.vn

1. INTRODUCTION

In the era of algorithmic proliferation and increasingly complex software ecosystems, the synthesis of source code via natural language interfaces has emerged as a critical axis of research at the confluence of formal language theory, neural representation learning, and automated reasoning [1], [2]. This convergence has revitalized longstanding questions in computability, expressivity, and syntactic alignment between human and machine representations of intent [3]. Traditional models of program synthesis, centered on formal grammars [4], software engineering principles [5], deductive synthesis, or enumerative search, have proven insufficiently scalable when confronted with the ambiguity and high dimensionality of natural language [6].

The advent of large-scale transformer-based language models (LLMs) [7], such as models in the GPT family [8], T5 [9], and code-specific models like CodeT5 [10] and StarCoder [11], has redefined the paradigm. By embedding symbolic structures into continuous vector spaces amenable to gradient-based

optimization, these models have achieved remarkable efficacy. However, they are often characterized by prohibitive parameterization (exceeding tens or hundreds of billions of weights), introducing challenges not only in terms of computational tractability and carbon footprint [12] but also epistemologically by obfuscating the interpretability and verifiability of generated code artifacts.

To address these limitations, small language models (SLMs) [13]–[15], typically constrained to sub-7B parameter regimes, have emerged as promising alternatives. While smaller, SLMs offer potential for efficient deployment on edge devices and within constrained inference environments. However, leveraging their full capability for domain-specific tasks like code generation requires effective adaptation. This study investigates the effectiveness of applying parameter-efficient fine-tuning (PEFT) techniques, specifically low-rank adaptation (LoRA) [16] and quantized low-rank adaptation (QLORA) [17], to prominent existing SLMs for code generation.

Our core hypothesis is that with efficient fine-tuning on a domain-specific dataset, these compact models can achieve performance comparable to, or even surpassing, larger baseline models, while requiring significantly fewer computational resources for training and deployment. We conduct empirical investigations on well-known SLMs including LLaMA 3 (1B and 3B variants) [18], Gemma 2B [19], and Qwen2.5 (1.5B and 3B variants) [20]. We fine-tune these models using LoRA/QLORA on the CodeAlpaca-20k dataset [21], a structured corpus designed for instruction-based code generation. We evaluate performance using the ROUGE-L metric and analyze the efficiency gains in terms of trainable parameters.

Our contributions are threefold: i) an empirical demonstration of the effectiveness of PEFT (LoRA/QLORA) for adapting existing SLMs to code generation, ii) a comparative analysis of several prominent SLMs under these fine-tuning regimes on the CodeAlpaca-20k benchmark, highlighting their relative strengths and limitations, and iii) evidence that efficiently fine-tuned sub-3B models can outperform larger baseline models on this task, suggesting the importance of effective adaptation over brute-force parameter scale. Ultimately, this work contributes to formalizing a scalable methodology for efficient neural code generation, suitable for both academic replication and real-world software development workflows.

2. RELATED WORK

Code generation using natural language prompts has evolved significantly with the emergence of large-scale pretrained models. Early models such as CodeBERT [22] and Code2Seq [23] relied heavily on syntactic features and were limited in their ability to generalize beyond predefined patterns or abstract syntax trees (ASTs). These models offered modest success in code retrieval and classification tasks but lacked the semantic compositionality and contextual understanding essential for realistic code synthesis. The paradigm shifted with the advent of transformer-based models pretrained on massive code corpora. Models such as CodeT5 [10], PolyCoder, Phi-3, Phi-3 Meets Law [15], [24], along with instruction-tuned variants like StarCoder [11] and WizardCoder [25], introduced architectural and objective-function refinements that greatly enhanced performance. Despite their improvements, these models typically exceed 6 billion parameters, posing significant challenges for deployment on constrained hardware and increasing computational costs.

To address these limitations, the research community has pivoted towards SLMs, which offer a more sustainable and accessible alternative [13], [14], [26]. This trend is exemplified by models like Google's Gemma 2B [19] and Alibaba's Qwen2.5 series [20], which leverage optimized Transformer architectures to deliver impressive performance within a sub-3B parameter footprint. These compact models are not only more efficient but also exhibit strong capabilities in multilingual and specialized tasks, making them promising candidates for domain-specific applications like code generation. While their potential is clear, their performance on specialized code generation benchmarks, particularly after targeted adaptation, remains an area requiring thorough investigation.

A key factor in unlocking the potential of these SLMs is the use of specialized datasets and instruction-tuning. Datasets like CodeAlpaca-20k [21], which provide a corpus of instruction-code pairs, are instrumental in teaching models to map natural language intent to syntactically correct and semantically appropriate code. By fine-tuning on such datasets, models learn to follow complex instructions, thereby moving beyond simple pattern matching to a more robust form of code synthesis. This methodology has become a cornerstone for adapting general-purpose language models to the nuanced domain of software engineering.

However, even for SLMs, full fine-tuning can be computationally prohibitive. This has led to the widespread adoption of parameter-efficient fine-tuning (PEFT) techniques. Methods like low-rank adaptation (LoRA) [27], which freezes pretrained model weights and injects trainable low-rank matrices, and quantized low-rank adaptation (QLORA) [17], which further reduces memory usage by quantizing the base model to 4-bits, have become instrumental. These techniques enable the adaptation of SLMs on consumer-grade

hardware while preserving or even enhancing performance on downstream tasks. Building on this line of research, our study evaluates the application of LoRA and QLORA on various SLMs such as Gemma 2B [19], Qwen2.5 [20], and smaller LLaMA 3 variants [18], focusing on standardized code generation benchmarks to assess performance and efficiency.

While previous works have explored fine-tuning individual SLMs for code [18], [20], a comprehensive, side-by-side comparative analysis of the leading SLMs from different major artificial intelligence (AI) providers (Google, Meta, Alibaba) under a unified PEFT framework is still lacking. Our work directly addresses this gap by systematically evaluating the performance of these models when fine-tuned with LoRA/QLORA on the CodeAlpaca-20k benchmark. This approach allows for a direct comparison of their inherent architectural strengths and their adaptability to the code generation domain, providing critical insights into the most effective and efficient pathways for democratizing neural code synthesis.

3. METHOD

In constructing a comprehensive experimental framework for evaluating the efficacy of small-scale transformer-based architectures in the context of source code generation, our methodology is predicated upon a multi-tiered approach that integrates: i) architectural selection under parameter constraint, ii) dataset curation and task formalization, iii) implementation of advanced fine-tuning regimes leveraging parameter-efficient adaptation, and iv) empirical validation via standardized lexical similarity metrics. This multipronged stratagem ensures both methodological rigor and reproducibility within constrained computational topologies.

3.1. Model selection under parameterized constraints

Let M denote the hypothesis space of autoregressive language models instantiated over a parameter domain $\Theta \subset \mathbb{R}^n$, with $n < 3 \times 10^9$. We select five representative models $M_i \in \mathsf{M}$, each characterized by architectural sparsity, multilingual capability, and decoder-only transformer backbones: M_1: Llama-3.2-1B-Instruct – 1B; M_2: Llama-3.2-1B-Instruct – 3B; M_3: Gemma – 2B; M_4: Qwen2.5 – 1.5B; M_5: Qwen2.5 – 3B. Each M_i is initialized with pretrained weights θ_i^0 and subjected to subsequent adaptation on a task specific distribution D_{code}.

Llama-3.2-1B-Instruct (M_1) is a 1B parameter language model from 3.2. It is designed with an optimized lightweight architecture. This model is well suited for deployment on personal and mobile devices, enhancing performance across various applications.

Llama-3.2-3B-Instruct (M_2) is a 3B-parameter language model from 3.2, designed to support multiple languages and optimized for tasks such as conversation, information retrieval, and text summarization. This powerful and flexible model is tailored for personal and mobile devices, delivering high efficiency in multilingual natural language processing.

Gemma-2-2b-it(M_3) is an open-source LLM developed by Google with 2 billion parameters. This model is designed for natural language processing while being optimized for resource-constrained environments like personal computers and mobile devices. With a decoder-only Transformer architecture and enhancements such as sliding window attention and soft cap, Gemma-2-2B-IT outperforms other open models of similar size. Additionally, it is built for seamless integration into developers' and researchers' workflows, supporting popular artificial intelligence (AI) frameworks like Hugging Face Transformers, JAX, PyTorch, and TensorFlow.

Qwen2.5-1.5B-Instruct (M_4) is a language model from the Qwen2.5 series, developed by the Qwen team. With approximately 1.54 billion parameters, it is specifically optimized for instruction-following tasks. This model supports multiple languages, including Vietnamese, and can process extended contexts of up to 128,000 tokens, enhancing its effectiveness in text generation, question answering, programming, and mathematical reasoning. Built on a Transformer architecture, it incorporates advanced techniques such as RoPE, SwiGLU, and RMSNorm to improve efficiency and accuracy. Qwen2.5-1.5B-Instruct is released under the Apache 2.0 license, enabling free usage and distribution across various applications.

Qwen2.5-3B-Instruct(M_5) is an advanced language model in the Qwen2.5 series, developed by the Qwen team. With approximately 3.09 billion parameters, it is fine-tuned specifically for instruction-following tasks. This model supports multiple languages, including Vietnamese, and can process context lengths of up to 128,000 tokens, enhancing its capabilities in text generation, question answering, programming, and mathematical problem solving. Built on a Transformer architecture, it incorporates optimization techniques such as RoPE, SwiGLU, and RMSNorm to enhance efficiency and accuracy. Qwen2.5-3B-Instruct is released under the Qwen Research license, permitting its use and distribution in research projects. With its ability to generate and understand high-quality text, this model serves as a powerful tool for natural language processing applications across diverse languages and contexts.

3.2. Dataset, formalization and preprocessing

Large language models (LLMs) excel at many tasks thanks to huge pretrained datasets. For our experiments, we use the sahil2801/CodeAlpaca-20k dataset [21]. Dataset CodeAlpaca-20k is a data set for training and evaluating natural language processing (NLP) models in the programming field. CodeAlpaca-20k is a dataset containing approximately 20,000 programming code samples and related comments. It is designed to support research and development of AI models capable of understanding and generating programming code.

The structure of this dataset includes an input data set described in natural language about the requirements of the programming problem; The output is a piece of programming code that performs the functions described. CodeAlpaca-20k can contain code from many different programming languages such as Python, JavaScript, Java, C++, and many others, depending on the goal of using the dataset.

We define a supervised dataset $D_{code} = \{(x_j, y_j)\}_{j=1}^{N}$, where $x_j \in \Sigma^*$ denotes a natural language task instruction and $y_j \in \Gamma^*$ represents the corresponding source code snippet. This dataset, CodeAlpaca-20k, is a structured corpus encompassing multi-lingual code representations across diverse programming paradigms P = {Python,C++,Java, ... }. To enforce syntactic uniformity and model compatibility, we implement a templated serialization thereby constraining tokenization under consistent positional embeddings.

$$T: (x_j, y_j) \mapsto \langle < |\text{user}| > x_j < |\text{end}| >< |\text{assistant}| > y_j < |\text{end}| > \rangle,$$

3.3. Constraints fine-tuning paradigms

3.3.1. Supervised fine-tuning (SFT)

Let $L_{\text{CE}}(\theta; x, y)$ denote the cross-entropy loss function parameterized by model weights θ, computed over the conditional likelihood $P_\theta(y \mid x)$. The supervised fine-tuning objective is defined as (1):

$$\theta^* = arg \min_{\theta} \sum_{j=1}^{N} L_{\text{CE}}(\theta; x_j, y_j) \quad (1)$$

Implementation is realized via the $HuggingFace\ SFTTrainer$ API, where gradient flow is restricted to selected.

3.3.2. Low-rank adaptation (LoRA)

To circumvent the infeasibility of full weight updates, we invoke $LoRA$, which approximates weight perturbations via a constrained low-rank subspace. Let $\Delta W \approx BA$, where $A \in \mathrm{R}^{r \times k}$, $B \in \mathrm{R}^{d \times r}$, and $r \ll min\ (d, k)$. The adapted weights are:

$$W = W_0 + \alpha \cdot BA \quad (2)$$

with α being a scaling hyperparameter. The fine-tuning is restricted to attention matrices ($q_{\text{proj}}, k_{\text{proj}}, v_{\text{proj}}, o_{\text{proj}}$) and feed-forward layers ($up, down, gate$).

3.3.3. Quantized low-rank adaptation (QLoRA)

For further compression, we adopt 4-bit $NormalFloat$ quantization ($nf4$), integrating it with LoRA adapters. We define the quantization mapping as (3):

$$W_{\text{quant}} = \text{round}\left(\frac{W - \min(W)}{\max(W) - \min(W)} \cdot (2^b - 1)\right), b = 4 \quad (3)$$

Fine-tuning language models for specialized domains like the code generation tasks requires significant computational resources. To mitigate this, we employed QLoRA [17], a technique designed to efficiently fine-tune LLMs while minimizing memory footprint and training time. QLoRA achieves this by quantizing the pre-trained model weights to a lower precision (4-bit) and then applying low-rank updates during the fine-tuning process. These low-rank updates are stored with higher precision (fp16), allowing for effective adaptation while keeping overall memory requirements low. Specifically, we utilized 4-bit quantization ($load - in - 4bit = True$) with the NF4 quantization type, striking a balance between model compression and performance. This configuration enabled us to fully leverage the capabilities of our chosen models while operating within practical resource constraints. This approach significantly accelerated the fine-tuning process without compromising the model's performance on the code generation tasks. This

transformation drastically reduces memory overhead while maintaining functional fidelity via a dequantization inverse mapping at inference.

3.4. Evaluation metric formalization

The primary quantitative instrument employed is ROUGE-L, a recall-oriented metric derived from the longest common subsequence (LCS) framework. Given a generated sequence $G = \{g_1, \dots, g_m\}$ and a reference sequence $R = \{r_1, \dots, r_n\}$, the LCS is denoted as $\text{LCS}(G, R)$. The corresponding ROUGE-L precision P , recall R , and F1-score F are computed as:

$$P = \frac{|\text{LCS}(G,R)|}{|G|}, R = \frac{|\text{LCS}(G,R)|}{|R|}, F = \frac{2PR}{P+R} \quad (4)$$

This metric is particularly suited for evaluating source code, as it implicitly captures both lexical coherence and structural adherence without penalizing minor syntactic permutations.

3.5. Experiment setup

All experiments were conducted on a server equipped with 4× NVIDIA RTX A5000 GPUs (24 GB VRAM each), dual AMD Ryzen Threadripper PRO 5965WX CPUs (48 logical cores), and 256 GB RAM. We used PyTorch 2.5.1 with CUDA 12.1, Hugging Face Transformers 4.46.3, and PEFT 0.13.2. Models were fine-tuned on the CodeAlpaca-20k dataset using LoRA and QLoRA. Training employed the AdamW optimizer with a learning rate of 2×10^{-4}, cosine scheduler, and early stopping based on validation loss. For LoRA, we set rank $r = 8$, scaling $\alpha = 16$, and dropout $= 0.05$. Table 1 summarizes the key training hyperparameters across different models. Figure 1 shows the computational resources and training environment.

Table 1. Training hyperparameters across models

Models	Params	Batch size	Epochs	Trainable Params	Steps
Llama-3.2-1B	1.0B	160	50	11.3M	750
Llama-3.2-3B	3.2B	64	50	24.3M	1950
Gemma-2-2B	2.0B	128	50	20.8M	950
Qwen2.5-1.5B	1.5B	128	50	18.5M	950
Qwen2.5-3B	3.1B	64	50	29.9M	1950

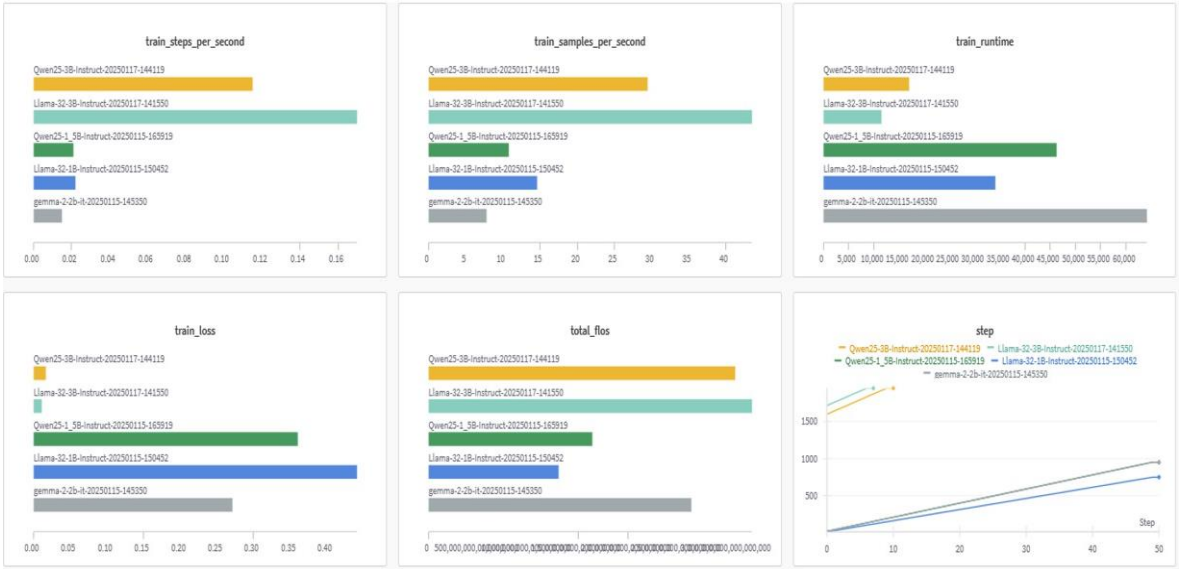

Figure 1. Computational resources and training environment

4. RESULTS AND DISCUSSION

To rigorously evaluate the generative efficacy of parameter-efficiently fine-tuned small language models within the domain of source code synthesis, we operationalize a multifaceted evaluation protocol

grounded in established textual similarity metrics, comparative benchmarking against relevant baselines and less-tuned versions of the SLMs, and controlled ablation analysis. As shown in Figure 2, our evaluation adheres to the principle of model-output congruence, in which the semantic fidelity of generated source code is quantified against human-curated reference implementations using ROUGE-L. Our experiments focused on comparing the performance gains achieved through fine-tuning these smaller models and analyzing their effectiveness relative to each other and to prior work utilizing larger language models. The findings of these experiments are summarized in Table 2. Prior to fine-tuning, the base models demonstrated limited proficiency in the domain.

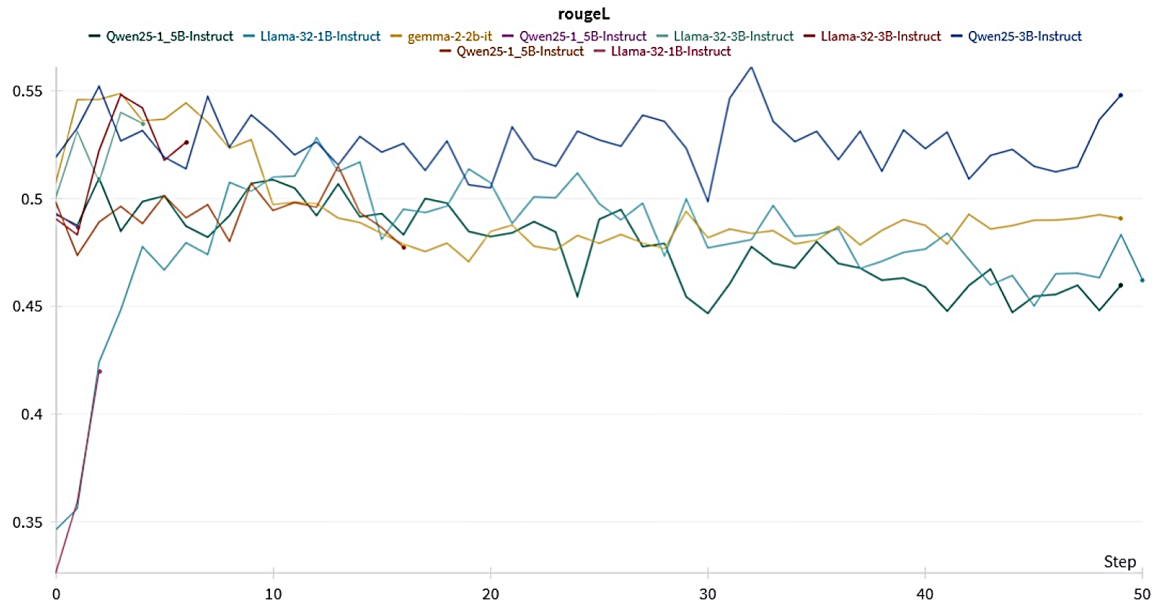

Figure 2. ROUGE-L performance of fine-tuned SLMs during training steps

4.1. Comparative performance synthesis

Let M_{base} denote the baseline model ensemble {CodeBERT,Code2seq,Phi-3 Mini 4K} , and let M_{fine} represent the set of fine-tuned small models studied in this work. The empirical results, summarized in Table 2, reveal a pronounced superiority of our models across all evaluated instances. The empirical results, summarized in Table 2, reveal a pronounced superiority of our fine-tuned SLMs (M_{fine}) across all evaluated instances compared to the M_{base} baselines and generally improved performance compared to M_{base}.

Table 2. Model comparison on ROUGE-L score

Model	Params	Trainable Params	Training Steps	ROUGE-L
CodeBERT	110M	110M	-	0.36
Code2seq	200M	200M	-	0.33
Phi-3 Mini 4K base	3.8B	3.8B	-	0.17
Our result				
Llama-3.2-1B (base)	1B	1B	-	0.45
Llama-3.2-1B-Instruct	1B	7M	4K	0.46
Llama-3.2-3B (base)	3.21B	3.21B	-	0.49
Llama-3.2-3B-Instruct	3.21B	15M	4K	0.54
Gemma-2-2b-it (base)	2B	2B	-	0.46
Gemma-2-2b-it-Instruct	2B	10M	4K	0.49
Qwen2.5-1.5B (base)	1.5B	1.5B	-	0.48
Qwen2.5-1.5B-Instruct	1.5B	7M	4K	0.46
Qwen2.5-3B (base)	3.1B	3.1B	-	0.51
Qwen2.5-3B-Instruct	3.1B	15M	4K	0.55

4.2. Ablation and interpretation

A critical juxtaposition is drawn between Phi-3 Mini 4K base (3.8B) and our fine-tuned 3B models (LLaMA 3 3B, Qwen2.5 3B), wherein the latter demonstrably outperform despite fewer total parameters

and significantly fewer trainable parameters during fine-tuning (15M vs 3.8B). This discrepancy substantiates the hypothesis that effective parameter-efficient fine-tuning and data alignment can be more impactful than brute-force base model size or full model fine-tuning for domain-specific tasks. The results suggest that it is not merely the volumetric size of a model that governs performance, but rather the synergistic interplay between training objective, data alignment, and efficient adaptation techniques like LoRA and QLoRA.

Furthermore, the consistent performance across the LLaMA-1B and Qwen-1.5B variants both yielding ROUGE-L=0.46 implies a potential capacity ceiling when using these fine-tuning techniques at lower parameter thresholds (1-1.5B), suggesting diminishing marginal returns without alternative adaptation strategies or potential architectural modifications to better suit the code domain at this size.

4.3. Performance comparison with earlier models: CodeBERT and Code2seq

CodeBERT and Code2seq were key stages in the development of neural code generation, although they mostly used syntactic representations and had trouble capturing deeper semantic linkages in source code. Their ROUGE-L scores of 0.36 and 0.33 show how limited they are, especially when it comes to activities that need strong semantic synthesis and awareness of context. In contrast, our finely tuned small language models (SLMs) consistently beat these baselines, with ROUGE-L scores as high as 0.55. This shows that parameter efficient fine-tuning methods not only make models work better but also greatly improve the quality of code generation.

4.4. Epistemological reflections

Beyond raw metrics, our findings gesture toward a broader epistemological implication: that effective adaptation and efficient architectures rather than brute-force parameter expansion may define the next frontier of neuro-symbolic code generation. These results advocate for a paradigm shift toward task-specific efficient fine-tuning protocols, enabling democratized deployment of capable SLMs without compromising output fidelity on domain tasks.

4.5. Limitations of the study

The evaluation of code generation models has several limitations: first, it relies solely on the ROUGE-L metric for lexical similarity, which does not assess functional correctness, such as compilability or output accuracy, necessitating future incorporation of execution-based benchmarks like HumanEval and MBPP for a more comprehensive assessment; second, the fine-tuning was limited to the CodeAlpaca-20k dataset, potentially leading to overfitting to its specific instruction styles and problem distributions, so testing on diverse datasets is essential for better generalization; third, hyperparameters for PEFT (*e.g.*, LoRA rank r=8, α=16, learning rate) were selected based on best practices without exhaustive optimization, suggesting that model-specific tuning via grid search or Bayesian optimization could enhance performance; and fourth, the evaluation used a static dataset, failing to reflect real-world interactive software development with multi-turn refinements, thus recommending exploration of conversational AI frameworks for handling follow-ups, corrections, and iterative improvements.

5. CONCLUSION AND FUTURE WORK

This paper presents a comparative study on parameter-efficient fine-tuning (PEFT) techniques specifically LoRA and QLoRA applied to several small language models (SLMs) including LLaMA 3.2, Qwen2.5, and Gemma. The models are fine-tuned on the CodeAlpaca-20k dataset for the task of code generation, and evaluated using ROUGE-L as the primary metric. The study demonstrates that fine-tuned SLMs can outperform much larger baseline models, highlighting their potential for low-resource deployment in software engineering contexts. The paper is well organized, methodologically sound, and provides clear empirical evidence supporting the effectiveness of PEFT in enhancing the performance of compact models. Overall, it is a relevant and timely contribution to the field of efficient neural code generation.

This work substantiates the potential of small, efficiently tuned models as viable, cost-effective, and sustainable alternatives to large, computationally demanding models for domain-specific software engineering problems. Their resource efficiency makes them particularly well-suited for deployment on edge and mobile devices, thus supporting the broader democratization of AI-assisted coding.

Future research directions stem directly from these promising results. We plan to explore the comparative efficacy of other PEFT methods and refine hyperparameter tuning for optimal performance. Expanding the training data with richer, interactive code-related conversations could enhance the models' ability to handle complex requests. Applying these techniques to SLMs for different programming domains, such as hardware description languages or smart contracts, represents another important avenue.

Additionally, investigating knowledge distillation from larger models and exploring architectural modifications specifically designed for code synthesis within the SLM paradigm are crucial steps towards developing even more capable and efficient neural code generators.

ACKNOWLEDGMENTS

This research was supported by the ĐH2025-TN07-07 project conducted at the Thai Nguyen University of Information and Communication Technology, Thai Nguyen, Vietnam, with additional support from the AI in Software Engineering Lab. The authors would like to thank the valuable feedback provided by the reviewers.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Van-Viet Nguyen	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
The-Vinh Nguyen	✓	✓			✓	✓		✓	✓	✓	✓	✓		✓
Huu-Khanh Nguyen				✓	✓	✓		✓	✓	✓				
Duc-Quang Vu			✓				✓		✓	✓		✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors state no conflict of interest. The authors have no financial, personal, or professional relationships that could inappropriately influence the research presented in this paper.

INFORMED CONSENT

We have obtained informed consent from all individuals included in this study.

ETHICAL APPROVAL

This research does not require ethical approval as it does not involve human participants, animal subjects, or sensitive data.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] E. N. Crothers, N. Japkowicz, and H. L. Viktor, "Machine-generated text: A comprehensive survey of threat models and detection methods," *IEEE Access*, vol. 11, pp. 70977–71002, 2023, doi: 10.1109/ACCESS.2023.3294090.
[2] Q. Zhang *et al.*, "A survey on large language models for software engineering," *arXiv preprint arXiv:2312.15223*, 2023.
[3] Y. Jernite *et al.*, "Data governance in the age of large-scale data-driven language technology," in *ACM International Conference Proceeding Series*, 2022, pp. 2206–2222, doi: 10.1145/3531146.3534637.
[4] S. Barke, E. A. Gonzalez, S. R. Kasibatla, T. Berg-Kirkpatrick, and N. Polikarpova, "HYSYNTH: Context-free LLM approximation for guiding program synthesis," in *Advances in Neural Information Processing Systems*, 2024, vol. 37, pp. 15612–15645.
[5] N. Van Viet and N. T. Vinh, "Large language models in software engineering," *Journal of Education For Sustainable Innovation*, vol. 2, no. 2, pp. 146–156, Dec. 2024, doi: 10.56916/jesi.v2i2.968.
[6] R. G. Dromey, "Formalizing the transition from requirements to design," in *Mathematical Frameworks For Component Software: Models For Analysis And Synthesis*, World Scientific, 2006, pp. 173–205.
[7] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural Information Processing Systems*, vol. 30, 2017.

[8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, 2019.
[9] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.
[10] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *EMNLP 2021 - 2021 Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2021, pp. 8696–8708, doi: 10.18653/v1/2021.emnlp-main.685.
[11] R. Li and others, "StarCoder: May the source be with you!," *arXiv preprint arXiv:2305.06161*, 2023.
[12] D. Patterson *et al.*, "Carbon emissions and large neural network training," *arXiv preprint arXiv:2104.10350*, 2021.
[13] P. Zhang, G. Zeng, T. Wang, and W. Lu, "TinyLlama: An open-source small language model," *arXiv preprint arXiv:2401.02385*. 2024.
[14] H. Wei *et al.*, "Small language model meets with reinforced vision vocabulary," *arXiv preprint arXiv:2401.12503*. 2024.
[15] M. Abdin, S. A. Jacobs, Y. Yang, and others, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2412.08905*. 2024.
[16] E. Hu *et al.*, "Lora: Low-rank adaptation of large language models," *ICLR 2022 - 10th International Conference on Learning Representations*, vol. 1, no. 2, p. 3, 2022.
[17] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLORA: Efficient finetuning of quantized LLMs," in *Advances in Neural Information Processing Systems*, 2023, vol. 36, pp. 10088–10115.
[18] H. Touvron and others, "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
[19] T. Lieberum *et al.*, "Gemma Scope: Open sparse autoencoders everywhere all at once on Gemma 2," in *BlackboxNLP 2024 - 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP - Proceedings of the Workshop*, 2024, pp. 278–300, doi: 10.18653/v1/2024.blackboxnlp-1.19.
[20] I. Ahmed *et al.*, "Qwen 2.5: A comprehensive review of the leading resource-efficient LLM with potential to surpass all competitors," *Authorea Preprints*. 2025, [Online]. Available: https://www.techrxiv.org/doi/pdf/10.36227/techrxiv.174060306.65738406/v1.
[21] S. Chaudhary, "Code alpaca: An instruction-following llama model for code generation," *GitHub repository*. 2023.
[22] Z. Feng *et al.*, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics Findings of ACL: EMNLP 2020*, 2020, pp. 1536–1547, doi: 10.18653/v1/2020.findings-emnlp.139.
[23] U. Alon, O. Levy, S. Brody, and E. Yahav, "Code2Seq: Generating sequences from structured representations of code," *arXiv preprint arXiv:1808.01400*, 2019.
[24] N. H. Khanh, V. N. Van, N. T. Vinh, and N. H. Cong, "Phi-3 meets law: Finetuning mini language models for legal document understanding," *Research, Development and Application on Information and Communication Technology|ISSN: 1859-3526*, vol. 2024, no. 3, pp. 136–142, 2024.
[25] Z. Luo *et al.*, "Wizardcoder: Empowering code large language models with evol-instruct," *arXiv preprint arXiv:2306.08568*, 2024.
[26] Y. Zhu, M. Zhu, N. Liu, Z. Xu, and Y. Peng, "Llava-phi: Efficient multi-modal assistant with small language model," in *EMCLR 2024 - Proceedings of the 1st International Workshop on Efficient Multimedia Computing under Limited Resources, Co-Located with: MM 2024*, 2024, pp. 18–22, doi: 10.1145/3688863.3689575.
[27] Z. Liu, J. Lyn, W. Zhu, and X. Tian, "ALoRA: Allocating low-rank adaptation for fine-tuning large language models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024, pp. 622–641.

BIOGRAPHIES OF AUTHORS

Van-Viet Nguyen is a researcher and Ph.D. student at the Thai Nguyen University of Information and Communication Technology, Thai Nguyen, Vietnam. He received a bachelor's in information technology at Thai Nguyen University (ICTU), Vietnam in 2009. He got a master's degree on Information Technology at Manuel S. Enverga University, Philippines in 2012. He researches interests include artificial intelligence, machine learning, and generative AI. He can be contacted at email: nvviet@ictu.edu.vn.

The-Vinh Nguyen is currently a senior lecturer at the Faculty of Information Technology, University of Information and Communication Technology. He graduated with a master's degree in information systems management from Oklahoma State University, USA (under scholarship 322). He completed his Ph.D. program under Project 911 in 2020 at Texas Tech University, USA. His main research interests are computer vision, computer visualization, and computer in human behavior. He has authored or coauthored more than 50 publications with 16 H-index and more than 850 citations. He can be contacted at email: vinhnt@ictu.edu.vn

Huu-Khanh Nguyen has graduated with a master's degree in computer science from the University of Information and Communications Technology - Thai Nguyen University since 2022 and is currently a PhD student here since 2023. His main research interests are computer science, natural language processing, generative AI and computer vision. He can be contacted at email: khanhnh@tnu.edu.vn

Duc-Quang Vu was born in Nam Dinh, Vietnam in 1991. He received a B.S. degree in education in information technology from the Thai Nguyen University of Education, Vietnam, in 2013 and an M.S. degree in information systems, from the University of Engineering and Technology (UET), Vietnam National University, Hanoi (VNU) in 2016. He received the Ph.D. degree in the Department of Computer Science and Information Engineering, National Central University, Taiwan in 2022 and a postdoc in 2023. His research interests include machine learning, deep learning, computer vision, speech processing, and bioinformatics. He can be contacted at email: vdquang@ictu.edu.vn.