Date of publication MAR-31, 2025, date of current version DEC-19, 2024.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.24.1.3885

Fine-Tuning Large Language Models for Code-Style Analysis: The Significance of Dataset Size

ANDRII HOLOVKO, VLADYSLAV ALIEKSIEIEV
Department of Applied Mathematics, Lviv Polytechnic National University, S. Bandery 12, Lviv, 79013, Ukraine
andrii.i.holovko@lpnu.ua, vladyslav.i.alieksieiev@lpnu.ua

Corresponding author: Andrii Holovko (e-mail: andrii.i.holovko@lpnu.ua).

ABSTRACT One aspect of a well-written codebase is its adherence to a particular code style, and Large Language Models (LLMs) can greatly assist in reviewing and adapting the code to follow the defined conventions. Because specific code-style rules are typically not known during the pre-training of the base model, additional fine-tuning is necessary. However, the exact number of training samples required to achieve optimal model performance is unclear. The significance of dataset size when fine-tuning LLMs to categorize Python code snippets as compliant or non-compliant with the specific PEP-8 indentation rule is investigated in this work. We used Low-Rank Adaptation (LoRA) and its quantized variant (QLoRA) to fine-tune the Llama 2 7B and Llama 3 8B models on datasets of varying sizes, ranging from 60 to 480 training samples. Our experiments demonstrated that the models fine-tuned with larger datasets (240 and 480 samples) achieved accuracies of up to 99%, whereas those trained with smaller datasets (60 and 120 samples) experienced overfitting and lower accuracy. Subsequent research will be based on these findings to explore the potential of LLMs and improve code readability, maintainability, and adherence to coding standards in software development. The methodology used to determine the sufficient number of training samples can also be valuable for fine-tuning LLMs in other domains where strict style or formatting conventions are required, such as legal document preparation, standardized medical reporting, or financial regulatory filings.

KEYWORDS code-style analysis; PEP-8; large language models; Llama 2; Llama 3; fine-tuning; dataset size; zero-shot learning; low-rank adaptation, QLoRA.

I. INTRODUCTION

Code-style analysis plays a vital role in software development by ensuring readability, maintainability, and consistency across codebases. Adhering to a uniform code style enhances collaboration between developers and reduces the likelihood of introducing errors, ultimately contributing to the overall quality and longevity of software projects.

Software quality and its evaluation have long been central concerns in software engineering research and practice. Established standards, such as ISO/IEC 25002:2024 [1], provide guidelines for assessing software quality, and extensive research has been conducted to develop methods for evaluating various software quality characteristics [2]. Code-style analysis methods commonly employ these standards, relying heavily on static analysis tools and manual reviews [3, 4]. Although these approaches can be effective, they are often time-consuming and prone to errors. Manual reviews are susceptible to oversight and inconsistencies; as human reviewers may overlook subtle code-style issues or interpret standards differently.

In contrast, Large Language Models (LLMs) are promising alternatives, leveraging their advanced language understanding capabilities to assist in various tasks, including code generation and reasoning. Models such as Llama 2 [5] and Llama 3 [6] have been successfully employed for diverse tasks, including natural language processing [7], sentiment analysis [8], and image recognition [9, 10].

Although LLMs are primarily known for their text-generation capabilities, several studies have explored their applications in classification tasks [11, 12], automated code reviews [13] and code understanding [14]. These studies have proven that fine-tuned LLMs can comprehend and generate programming code. However, the models used in these experiments are typically trained or fine-tuned on large datasets containing a significant amount of high-quality training data.

Another study found that the efficiency of fine-tuning is highly task- and data dependent [15]. The authors discovered that LLM fine-tuning follows a power-based multiplicative joint scaling rule between the fine-tuning data size and the other scaling factors. The findings of this study also indicate that the result of the fine-tuning process is affected more by the increase in the size of the LLM model than by the scaling of the pre-training data.

We began our study on how well LLMs work for code-style analysis by fine-tuning the pre-trained Llama 2 7B model to classify Python code snippets for compliance with the following three PEP-8 indentation rules: E101 (indentation contains mixed spaces and tabs), E111 (indentation is not a multiple of four), and E112 (expected an indented block) [16]. For each rule, 20 "compliant" and 20 "non-compliant" training examples were prepared. However, we faced significant challenges owing to the unstable training process, which led to inconsistent results in model performance. Our hypothesis implies that an insufficient amount of training data for each rule is the primary factor contributing to observed issues.

Motivated by the need to develop a model for our code-style analysis task, we set out to determine the minimum number of training samples required to produce reasonably good results. By examining the performance of models trained on datasets of varying sizes, we seek to understand how the training efficiency, model generalization, and accuracy are influenced by the amount of data available for fine-tuning.

The goal of this research is to enhance our understanding of fine-tuning LLMs for domain-specific tasks, specifically code-style analysis, and to evaluate the significance of the dataset size in achieving robust and reliable performance. Through this study, we aim to provide insights that can inform future research and practical applications of LLMs to improve software development, as well as similar tasks where enforcing a particular style or standard is vital.

II. METHODOLOGY

A. DATASET

The dataset used for fine-tuning in this study comprises 600 Python code snippets, balanced between "compliant" and "non-compliant" examples according to the PEP-8 style rule, which specifies that indentation must be a multiple of four (E111 in *pycodestyle* error codes taxonomy).

This dataset was derived from the "python code instructions 18k alpaca" dataset [17] and validated using the *pycodestyle* checker [18]. We made it publicly accessible on the Hugging Face hub [19].

The dataset was split into training and validation sets in an 80-20% proportion, with 480 records used for training and 120 for validation. To evaluate the impact of the dataset size on the fine-tuning process, the dataset was further split into subsets with 60 training and 15 validation samples, 120 training and 30 validation samples, and 240 training and 60 validation samples.

B. LOW-RANK ADAPTATION

All data processing and evaluation experiments were performed on an Apple MacBook M3 Max with 48GB of unified memory.

Training was conducted on a single NVIDIA A100-SXM4-40GB GPU system. We used the AdamW optimizer [20] and trained the models for up to five epochs using parameter-efficient fine-tuning techniques such as Low-Rank Adaptation (LoRA) [21] and QLoRA [22].

During the early training sessions, we constantly encountered out-of-memory issues, even with LoRA. To mitigate this problem, we employed quantization using the BitsAndBytes library [23]. This approach enabled us to fine-tune the model regardless of the batch size used, with a memory consumption not exceeding 25GB of GPU memory. However, BitsAndBytes does not currently support Apple silicon, which is why we used the NVIDIA GPU platform.

On the other hand, the MLX library [24] shows promise for leveraging Apple's GPU while supporting quantization. However, it is still under active development, and its functionality is yet to be fully explored.

LoRA was designed to adapt a pre-trained model to better suit a specific, often smaller, dataset by adjusting only a small subset of the model's weight parameters. In simpler terms, instead of fine-tuning all parameters of a large model, LoRA only updates a small percentage of them, reducing computational cost and required resources while preserving performance. This method is particularly valuable because it enables efficient fine-tuning of large models on task-specific data [25, pp. 317–319].

Usually, when deep neural networks are being fine-tuned, backpropagation learns a matrix ΔW, which indicates how much to update the original weight parameters to minimize the loss function during training (Fig. 1).

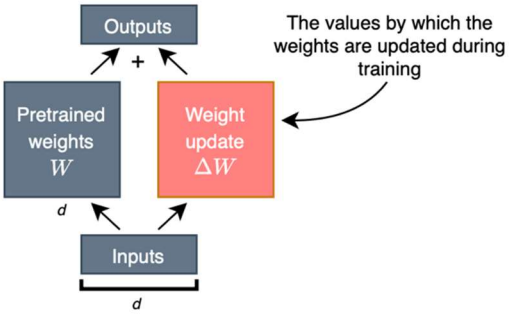

Figure 1. Weight update in regular fine-tuning

LoRA provides a more efficient alternative by approximating the weight updates ΔW with the product of two smaller matrices, A and B:

$$\Delta W \approx AB, \tag{1}$$

where A and B are much smaller than W, and AB represents their matrix multiplication product.

Using LoRA, the weight update can then be formulated as:

$$W_{updated} = W + AB. \tag{2}$$

Fig. 2 illustrates the operational approach of LoRA, employing matrices A and B of reduced dimensions to approximate ΔW, with rank r representing the adjustable hyperparameter.

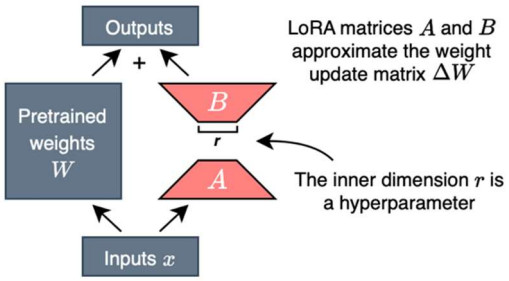

Figure 2. Weight update in LoRA

This approach allows for significant reductions in the number of parameters that need to be fine-tuned (Table 1), leading to faster training times and lower resource requirements, while still effectively adapting the model to the new data.

Table 1. Trainable parameters after applying LoRA

Model	Model Params	Trainable Params	Trainable %
Llama 2 7B	~6.7B	~2.1M	0.0319
Llama 3 8B	~7.5B	~5.1M	0.0682

QLoRA extends LoRA by quantizing the model's weights. This further reduces the memory footprint and can speed up training, making it more feasible to fine-tune large models on smaller hardware setups.

C. CLASSIFICATION HEAD

The input code snippet S was tokenized using *LlamaTokenizer* for Llama 2 and *AutoTokenizer* for Llama 3 from the Hugging Face transformers library [26].

The resulting tokens T were then input into the pre-trained or fine-tuned model to extract the latent representation H using *LlamaForSequenceClassification* transformer [27]:

$$T = Tokenizer(S)$$
$$H = LlamaForSequenceClassification(T). \quad (3)$$

LlamaForSequenceClassification uses the last token to perform classification. This token effectively captures all historical information due to the unidirectional information flow imposed by the causal masks. The vector representation was then passed through the fully connected layers and a *softmax* layer, mapping it to the label space.

Finally, the cross-entropy loss is computed using the output logits and the class labels (1 for "compliant" and 0 for "non-compliant" code sample).

D. EXPERIMENTS

The baseline performance of the pre-trained Llama 2 7B and Llama 3 8B models was evaluated using zero-shot classification methods [28, 29]. This step aimed to establish the ability of pre-trained models to differentiate between code-style compliance and non-compliance without any task-specific adjustments.

Following this, both the Llama 2 and Llama 3 base models underwent a series of fine-tuning experiments. Twenty experiments were conducted by systematically varying the number of epochs, batch sizes, and LoRA parameters. The dataset for these experiments comprised 120 training and 30 validation samples. We employed a Bayesian search approach [30] to identify the optimal hyperparameter configurations efficiently.

The hyperparameters of the top-performing Llama 2 and Llama 3 models were selected for further experiments. These optimized parameters formed the basis for exploring the accuracy of fine-tuned LLMs for code-style analysis across different dataset sizes.

The experiments were conducted on datasets with the following sizes: 60/15, 120/30, 240/60, and 480/120 training/validation samples.

Finally, the fine-tuned models were compared with the pre-trained models on the full dataset to evaluate improvements in code-style analysis accuracy after the fine-tuning process.

III. RESULTS

The source code for all experiments can be found in [31].

A. ZERO-SHOT CLASSIFICATION

The performance of pre-trained Llama 2 and Llama 3 models was evaluated through zero-shot binary classification across 10 experiments using the dataset's "validation" split (Fig. 3).

For Llama 2, the accuracy ranged from 0.4583 to 0.5833, with a mean accuracy of 0.5 and a standard deviation of 0.0350, indicating relatively moderate variability in performance. The highest accuracy was observed in Experiment 8, and the lowest in Experiments 1 and 9.

For Llama 3, the accuracy ranged from 0.4083 to 0.6667, with a mean accuracy of 0.4983 and a standard deviation of 0.0694, demonstrating slightly more variability in performance. The highest accuracy was observed in Experiment 3, and the lowest in Experiment 10.

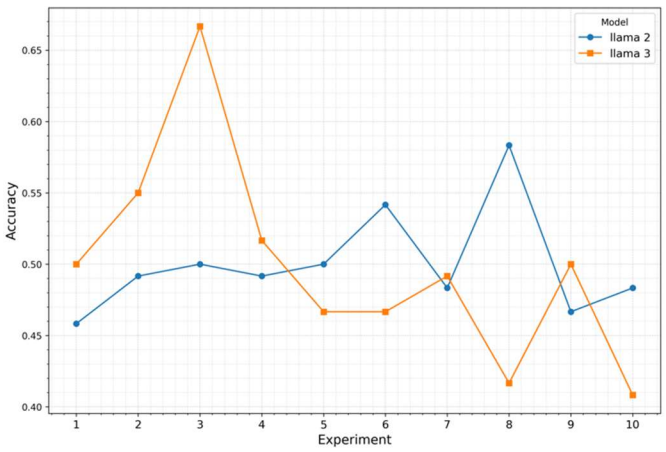

Figure 3. Zero-shot classification for pre-trained models

Given that this is a binary classification task, the results are close to those of random guessing (50% accuracy), indicating that neither model significantly outperforms random chance in this zero-shot setting. These results suggest that further fine-tuning or additional data may be necessary to improve the performance of models beyond random guessing.

B. HYPERPARAMETERS DISCOVERY

The hyperparameters tuning process utilized a Bayesian optimization approach to identify the optimal settings for training the models.

The hyperparameters considered were the number of epochs, batch size, and LoRA parameters (alpha, dropout, and rank). The evaluation metric used for optimization was the accuracy. Figures 4 and 5 illustrate the results of discovering the optimal hyperparameters for fine-tuning the Llama 2 and Llama 3 models, respectively.

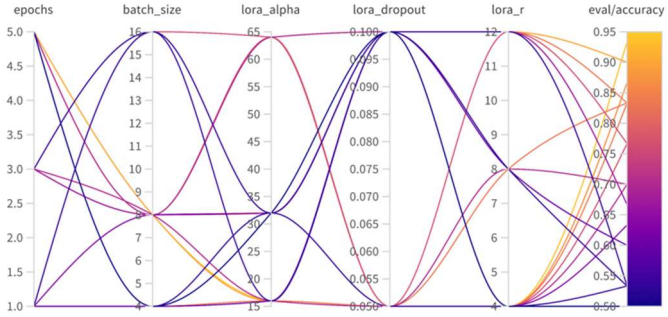

Figure 4. Tuning hyperparameters for Llama 2

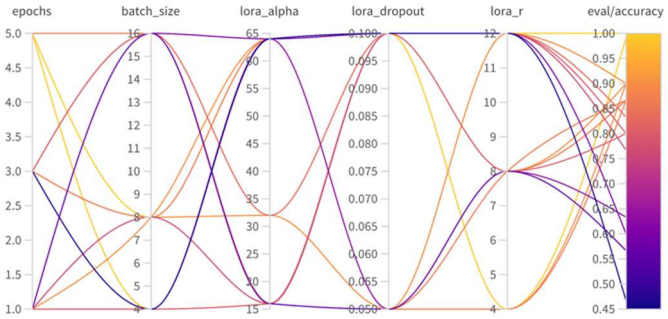

Figure 5. Tuning hyperparameters for Llama 3

The Bayesian optimization results indicated that smaller batch sizes (4 and 8) consistently yielded a higher evaluation accuracy, suggesting that frequent updates with smaller gradient steps are advantageous. Lower LoRA alpha values (16 for Llama 2 and up to 32 for Llama 3) and moderate LoRA dropout values (0.05) were prevalent in the high-performing configurations, highlighting the effectiveness of balanced regularization. In addition, configurations with a greater number of epochs demonstrated improved performance, underscoring the importance of extended training periods for model learning.

The optimal hyperparameters for fine-tuning Llama 2 and Llama 3 are presented in Table 2.

Table 2. Optimal hyperparameters for fine-tuning models

Parameter	Llama 2	Llama 3
epochs	5	5
batch_size	8	8
lora_alpha	16	32
lora_dropout	0.05	0.05
lora_r	4	12
eval/accuracy	**0.933**	**1**

We used a linear learning rate scheduler [32], with the learning rate set to 0.001 and the weight decay set to 0.01.

C. FINE-TUNING LLAMA 2 7B MODEL

We evaluated the impact of dataset size on fine-tuning the Llama 2 7B model by analyzing the training loss, validation loss, and obtained accuracy during the training process.

For the smallest dataset of 60 training samples and 15 validation samples, the training loss decreased rapidly (Fig. 6), reaching nearly zero by the fifth epoch. However, the validation loss increased after the first epoch, indicating overfitting (Fig. 7). Despite this, the accuracy improved slightly from 66.67% to 73.33%, showing limited generalization due to the small dataset size (Fig. 8).

For the 120/30 dataset, the training loss decreased significantly, and the validation loss showed a downward trend, dropping from 2.549609 to 1.103788. The accuracy improved considerably, reaching 83.33% by the fifth epoch, indicating a better generalization with a moderately larger dataset.

For the 240/60 dataset, the training loss decreased rapidly, and the validation loss decreased substantially from 0.707462 to 0.300284. The accuracy peaked at 96.67%, indicating robust model performance and strong generalization for this dataset size.

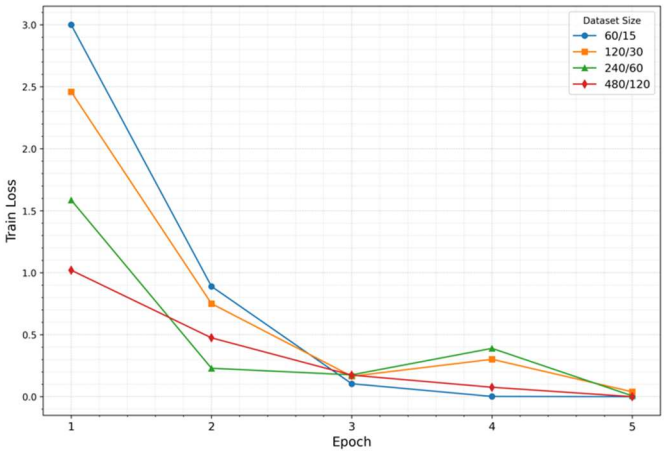

Figure 6. Training loss of Llama 2 in 5 epochs

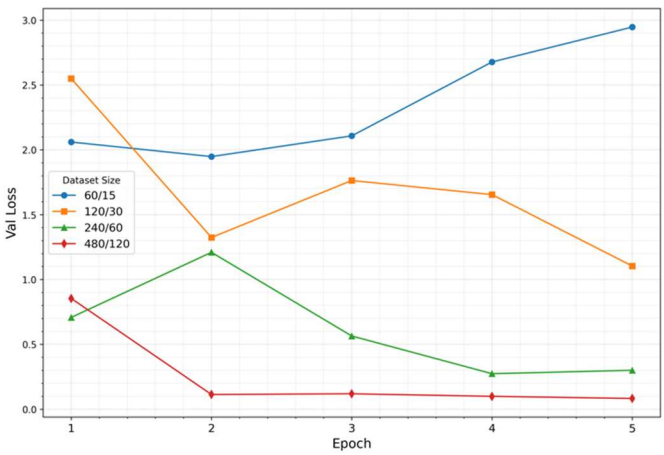

Figure 7. Validation loss of Llama 2 in 5 epochs

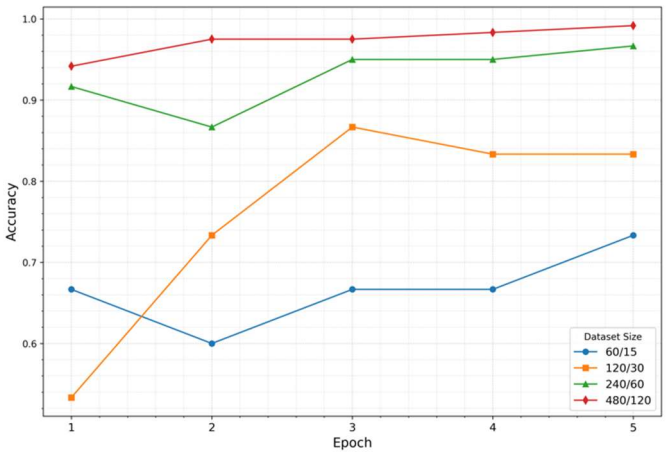

Figure 8. Accuracy of Llama 2 in 5 epochs

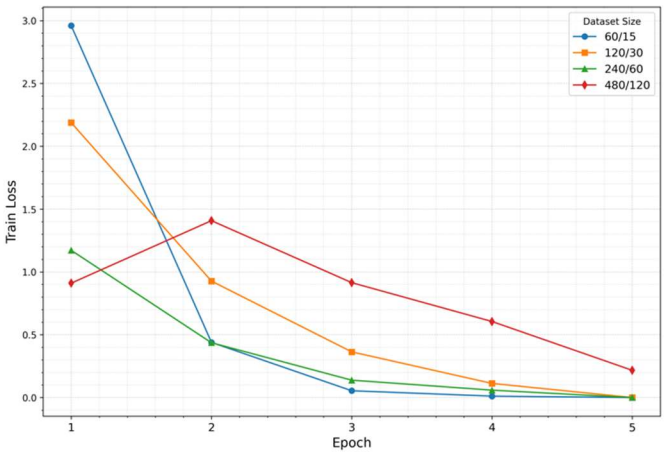

Figure 9. Training loss of Llama 3 in 5 epochs

The largest dataset, comprising 480 training samples and 120 validation samples, yielded the best results. The training loss steadily decreased to zero, whereas the validation loss continuously decreased to 0.082847. The accuracy reached an impressive 99.17%, demonstrating an excellent generalization.

It is important to note that instability in training (e.g., validation loss fluctuations) was more evident with smaller datasets, suggesting that a limited number of samples can impede the fine-tuning process by not providing a representative range of patterns for the model to learn.

Overall, larger datasets consistently led to lower training and validation losses and higher accuracy, highlighting the critical role of extensive training data in enhancing model performance and generalization for code-style classification tasks.

D. FINE-TUNING LLAMA 3 8B MODEL

For the smallest dataset (60/15), the training loss for Llama 3 decreased rapidly (Fig. 9), similar to that of Llama 2. However, the validation loss showed fluctuations, with an increase in the third epoch before decreasing again (Fig. 10). The accuracy improved from 66.67% to 86.67%, indicating better generalization than for Llama 2 for this dataset size (Fig. 11).

For the 120/30 dataset, the training loss significantly decreased. The validation loss initially decreased, but then slightly increased in the middle epochs before improving again. The accuracy reached 93.33%, showing better generalization compared with Llama 2 for this dataset size.

For the 240/60 dataset, the training loss decreased steadily, but the validation loss showed more fluctuation than that of Llama 2 model. Despite this, the accuracy peaked at 96.67%, indicating a similar performance to Llama 2.

For the largest dataset (480/120), the training loss showed an unusual pattern, with an increase in the second epoch before decreasing again. The validation loss initially decreased but then increased again in the second and third epochs. The accuracy reached 97.50%, indicating strong generalization but some instability in training.

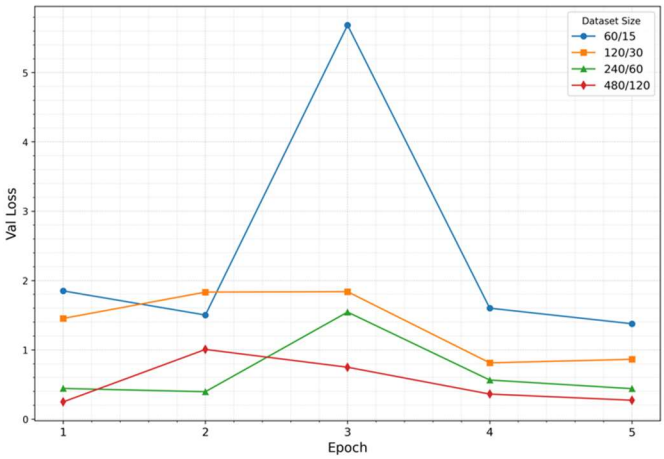

Figure 10. Validation loss of Llama 3 in 5 epochs

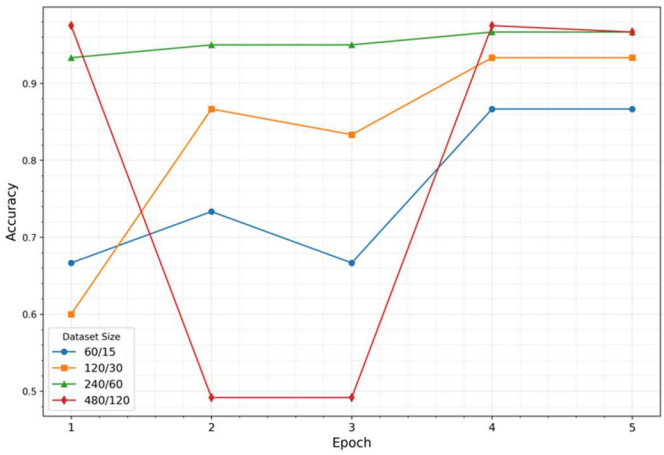

Figure 11. Accuracy of Llama 3 in 5 epochs

Overall, Llama 3 demonstrated slightly better accuracy for smaller datasets, but showed instability in training loss and validation loss for larger datasets. By contrast, Llama 2 exhibited more stable training and validation patterns, suggesting better consistency in fine-tuning with larger datasets. These fluctuations for Llama 3 indicate that even larger models can experience instability when the dataset is too small, possibly due to the model's capacity exceeding the complexity of the provided training data.

E. EVALUATION OF FINE-TUNED MODELS

Models fine-tuned with different dataset sizes were evaluated using 1200 previously unseen code samples [33]. The results are presented in Table 3.

Table 3. Accuracy of fine-tuned vs. pre-trained (base) models

Model		Accuracy %
Llama 2 7B	base	47.3
	fine-tuned (60/15)	77.6 (+30.3)
	fine-tuned (120/30)	88.6 (+41.3)
	fine-tuned (240/60)	98.4 (+51.1)
	fine-tuned (480/120)	99.5 (+52.2)
Llama 3 8B	base	45.2
	fine-tuned (60/15)	87.8 (+42.6)
	fine-tuned (120/30)	95.0 (+49.8)
	fine-tuned (240/60)	98.5 (+53.3)
	fine-tuned (480/120)	99.0 (+53.8)

The results demonstrate that fine-tuning on progressively larger datasets leads to a substantial increase in accuracy. Llama 3 performs better at smaller to mid-range dataset sizes, whereas Llama 2 achieves a slightly higher accuracy with the largest dataset. The most significant improvements for both models were observed with dataset sizes up to 240/60, where the accuracy gains were substantial, while beyond this point, the increase in accuracy became marginal. This indicates that while larger datasets generally enhance performance, the benefits plateau beyond a certain point.

IV. CONCLUSIONS

The primary aim of our research was to explore the impact of dataset size on fine-tuning LLMs for a particular code-style analysis task. We assessed how datasets of varying sizes used for fine-tuning impact the performance and reliability of models to categorize Python code as compliant or non-compliant with a specific PEP-8 indentation rule.

Our experiments demonstrated that models fine-tuned with larger datasets, specifically with 240 and 480 samples for a single rule, achieved accuracies of up to 99%. In contrast, the models trained with smaller datasets (60 and 120 samples) experienced overfitting and lower accuracy. These results suggest that having only dozens of samples per rule is insufficient for developing high-performance models; at least hundreds of training samples per rule are necessary. Our findings also show that insufficient data can lead to unstable training and suboptimal convergence, especially when fine-tuning large models.

We acknowledge the limitations of this study. We focused on only one PEP-8 rule; therefore, it is unclear how the models would perform with multiple rules. Additionally, we used only smaller LLM variants and did not fully explore the impact of quantization on the model quality.

Future research should address these limitations by investigating various PEP-8 rules, exploring larger LLM variants, experimenting with different fine-tuning strategies, and assessing the impact of quantization.

Beyond software development, these observations can be applied to other domains where compliance with formatting or stylistic standards is necessary, such as legal documentation, medical record keeping, and financial reporting.

Looking ahead, we plan to expand our research on Large Language Models to further explore their potential for enhancing software quality.

V. ACKNOWLEDGEMENTS

We thank Taras Kutsyk, an M.S. student in the Department of Applied Mathematics at Lviv Polytechnic National University, for his work on the dataset used in our early experiments on fine-tuning the Llama 2 model.

References

[1] *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model overview and usage*, ISO/IEC 25002:2024, Mar. 2024.

[2] T. Hovorushchenko and O. Pomorova, "Evaluation of mutual influences of software quality characteristics based ISO 25010:2011," *Proceedings of the 2016 IEEE XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT)*, Lviv, Ukraine, Sep. 2016, pp. 80–83. https://doi.org/10.1109/STC-CSIT.2016.7589874.

[3] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: a case study at google," in *Proceedings of the ACM 40th International Conference on Software Engineering: Software Engineering in Practice*, Gothenburg, Sweden, May 2018, pp. 181–190. https://doi.org/10.1145/3183519.3183525.

[4] S. Panichella, V. Arnaoudova, M. Di Penta, and G. Antoniol, "Would static analysis tools help developers with code reviews?," in *Proceedings of the IEEE 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Montreal, QC, Canada, Mar. 2015, pp. 161–170. doi: 10.1109/SANER.2015.7081826.

[5] H. Touvron *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models," Jul. 19, 2023, *arXiv*: arXiv:2307.09288. https://doi.org/10.48550/arXiv.2307.09288.

[6] AI@Meta, "Llama 3 Model Card." [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[7] B. M. Pavlyshenko, "Analysis of Disinformation and Fake News Detection Using Fine-Tuned Large Language Model," Sep. 09, 2023, *arXiv*: arXiv:2309.04704. https://doi.org/10.48550/arXiv.2309.04704.

[8] X. Zhu, S. Gardiner, T. Roldán, and D. Rossouw, "The Model Arena for Cross-lingual Sentiment Analysis: A Comparative Study in the Era of Large Language Models," Jun. 27, 2024, *arXiv*: arXiv:2406.19358. https://doi.org/10.48550/arXiv.2406.19358.

[9] J. Wang *et al.*, "Adapting LLaMA Decoder to Vision Transformer," May 27, 2024, *arXiv*: arXiv:2404.06773. https://doi.org/10.48550/arXiv.2404.06773.

[10] X. Chu, J. Su, B. Zhang, and C. Shen, "VisionLLaMA: A Unified LLaMA Backbone for Vision Tasks," Jul. 07, 2024, *arXiv*: arXiv:2403.00522. https://doi.org/10.48550/arXiv.2403.00522.

[11] Z. Li *et al.*, "Label Supervised LLaMA Finetuning," Oct. 02, 2023, *arXiv*: arXiv:2310.01208. https://doi.org/10.48550/arXiv.2310.01208.

[12] A. Jafari, "Comparison Study Between Token Classification and Sequence Classification in Text Classification," Nov. 25, 2022, *arXiv*: arXiv:2211.13899. https://doi.org/10.48550/arXiv.2211.13899.

[13] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, "LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning," Sep. 04, 2023, *arXiv*: arXiv:2308.11148. https://doi.org/10.48550/arXiv.2308.11148.

[14] B. Rozière *et al.*, "Code Llama: Open Foundation Models for Code," Jan. 31, 2024, *arXiv*: arXiv:2308.12950. https://doi.org/10.48550/arXiv.2308.12950.

[15] B. Zhang, Z. Liu, C. Cherry, and O. Firat, "When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method," Feb. 26, 2024, *arXiv*: arXiv:2402.17193. https://doi.org/10.48550/arXiv.2402.17193.

[16] "Pycodestyle Error Codes." [Online]. Available: https://pycodestyle.pycqa.org/en/latest/intro.html#error-codes

[17] "Python Code Instructions 18K Alpaca Dataset." [Online]. Available: https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca

[18] *pycodestyle*. Python Code Quality Authority. [Online]. Available: https://github.com/PyCQA/pycodestyle

[19] A. Holovko, "Dataset: PEP8 E111 Compliance." 2024. [Online]. Available: https://huggingface.co/datasets/aholovko/pep8_e111_compliance

[20] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," Jan. 04, 2019, *arXiv*: arXiv:1711.05101. https://doi.org/10.48550/arXiv.1711.05101.

[21] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," Oct. 16, 2021, *arXiv*: arXiv:2106.09685. https://doi.org/10.48550/arXiv.2106.09685.

[22] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," May 23, 2023, *arXiv*: arXiv:2305.14314. https://doi.org/10.48550/arXiv.2305.14314.

[23] T. Dettmers, *BitsAndBytes*. (2024). [Online]. Available: https://github.com/TimDettmers/bitsandbytes

[24] A. Hannun, J. Digani, A. Katharopoulos, and R. Collobert, *MLX: Efficient and flexible machine learning on Apple silicon*. (2023). [Online]. Available at: https://github.com/ml-explore

[25] S. Raschka, *Build a Large Language Model (From Scratch)*, MEAP v7. Manning Publications Co, 2024.

[26] *Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX*. Hugging Face. [Online]. Available at: https://github.com/huggingface/transformers

[27] *Llama for Sequence Classification*. Hugging Face. [Online]. Available at: https://github.com/huggingface/transformers/blob/63fb253df0d976b95d9b4b9a7b0012e5f8a37896/src/transformers/models/llama/modeling_llama.py#L1312

[28] "Zero-Shot Learning in Modern NLP," Joe Davison Blog. [Online]. Available at: https://joeddav.github.io/blog/2020/05/29/ZSL.html

[29] W. Yin, J. Hay, and D. Roth, "Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach," Aug. 31, 2019, *arXiv*: arXiv:1909.00161. https://doi.org/10.48550/arXiv.1909.00161.

[30] R. Turner *et al.*, "Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020," Aug. 31, 2021, *arXiv*: arXiv:2104.10201. https://doi.org/10.48550/arXiv.2104.10201.

[31] A. Holovko, *Fine-Tuning LLM for Code Style Analysis in Python*. (2024). Python. [Online]. Available at: https://github.com/aholovko/pycodestyle-llm

[32] A. Defazio, A. Cutkosky, H. Mehta, and K. Mishchenko, "When, Why and How Much? Adaptive Learning Rate Scheduling by Refinement," Oct. 11, 2023, *arXiv*: arXiv:2310.07831. https://doi.org/10.48550/arXiv.2310.07831.

[33] A. Holovko, "Dataset: PEP8 E111 Compliance Evaluation." 2024. [Online]. Available: https://huggingface.co/datasets/aholovko/pep8_e111_compliance_eval

***ANDRII HOLOVKO** received an M.S. in Social Informatics from Lviv Polytechnic National University, Ukraine in 2006, a Ph.D. student since 2023 at the Department of Applied Mathematics at Lviv Polytechnic National University, Ukraine. Since 2007, Andrii has been actively engaged in the software development industry. His academic pursuits are focused on bridging the gap between his industry experience and research interests in Artificial Intelligence and Machine Learning*

***VLADYSLAV ALIEKSIEIEV** received the Ph.D. degree in Technical Science (Mathematical Modelling and Computational Methods) in 2009, after graduating Lviv Polytechnic National University in 2002 as M.S. in Applied Mathematics. Holds the position of Associate Professor at the Department of Applied Mathematics in Lviv Polytechnic National University since 2012. Current areas of scientific interests include AI, ML, Data Science, Mathematical Modelling.*