LLMs Fine-Tuning Report

Abstract

A detailed report on the process and results of fine-tuning our off-the-shelf LLMs for code completion on Python security code and React 18.2.0 code. The report includes the dataset collection pipeline, both datasets statistics, fine-tuning details, fine-tuned model evaluation, benchmarking, and code samples. The PDF version of the report can be found in the code repository. The API to use the models and training package will be released in the coming milestones.

1 Dataset Collection Pipeline

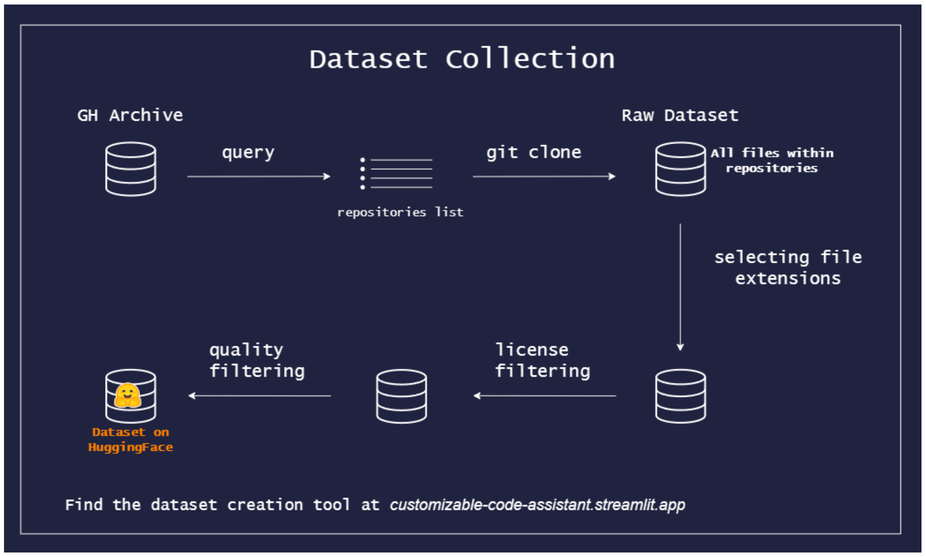
Dataset Collection
GH Archive
query
repositories list
git clone
Raw Dataset
All files within repositories
selecting file extensions
license filtering
quality filtering
Dataset on HuggingFace
Find the dataset creation tool at customizable-code-assistant.streamlit.app

Figure 1: Our Dataset Collection Pipeline (Inspired by TheStack by BigCode)

2 Datasets Statistics

2.1 React Dataset Statistics

- Number of Repositories: 22
- Number of Files: 2046
- File Extensions: { "js" }
- Average Code Snippet Length: 7261.95
- Average Code Snippet Lines: 238.76
- Average Code Snippet Line Length: 37.39
- Average Code Snippet Alphanumeric Ratio: 0.43

Python-React-Code-Dataset: `https://huggingface.co/datasets/ammarnasr/Python-React-Code-Dataset`

2.2 Security Dataset Statistics

- Number of Repositories: 30
- Number of Files: 1598
- File Extensions: { "py" }
- Average Code Snippet Length: 4453.23
- Average Code Snippet Lines: 111.88
- Average Code Snippet Line Length: 38.81
- Average Code Snippet Alphanumeric Ratio: 0.60

Python-Security-Code-Dataset: `https://huggingface.co/datasets/ammarnasr/Python-Security-Code-Dataset`

3 Fine-Tuning Details

3.1 General Information

- Base Model: Salesforce - CodeGen2 1B
- Description: CodeGen2 is a family of autoregressive language models for program synthesis, introduced in the paper:CodeGen2: Lessons for Training LLMs on Programming and Natural Languages by Erik Nijkamp*, Hiroaki Hayashi*, Caiming Xiong, Silvio Savarese, Yingbo Zhou. CodeGen2 is capable of infilling, and supports more programming languages. Four model sizes are available: 1B, 3.7B, 7B, 16B.
- Total Number of Parameters: 1.1×10^9

3.2 Fine-Tuning Technique

- Fine-Tuning Technique: Low Rank Adaptation (LoRa)
- Description: LoRA is a fine-tuning technique for LLMs that can be used to adapt LLMs to new tasks and domains with limited data. LoRA is one of the efficient fine-tuning techniques that can be used to fine-tune LLMs known as Parameter-Efficient Fine-Tuning (PEFT)..
- Number of LoRa Parameters: 2.2×10^6
- Hardware: 1x Tesla V100

3.3 React Dataset Fine-Tuning Details

- Tokens per Epoch: 1.35×10^6
- Expected Loss: 1.096
- Number of Epochs: 10
- Fine-Tuning Time (Hours): 12.5
- Fine-Tuning Task: Autoregressive Code Completion
- Fine-Tuning Domain: React 18.2.0
- Minimum Validation Loss: 0.973496293

Python-React-Code-Model: `https://huggingface.co/ammarnasr/codegen2-1B-react`

3.4 Security Dataset Fine-Tuning Details

- Tokens per Epoch: 0.63×10^6
- Expected Loss: 1.126
- Number of Epochs: 10
- Fine-Tuning Time (Hours): 8.43
- Fine-Tuning Task: Autoregressive Code Completion
- Fine-Tuning Domain: Python Security Code
- Minimum Validation Loss: 1.082985498

Python-Security-Code-Model: `https://huggingface.co/ammarnasr/codegen2-1B-security`

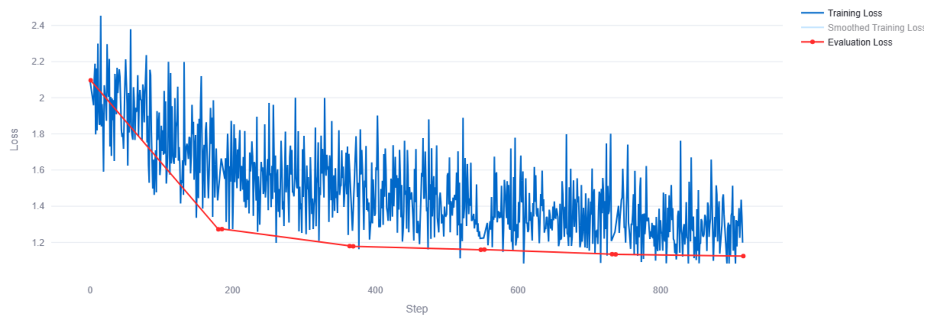

Figure 2: Training and Evaluation Loss for the Security Dataset

4 Fine-Tuned Models Evaluation

4.1 Loss Plot Snippet

4.2 Perplexity and Performance Benchmarking

4.2.1 Perplexity Details

- Benchmark: Perplexity
- Description: Perplexity is a metric used to evaluate the performance of language models. The lower the perplexity, the better the model. The Benchmarking is done on the test set, an independent set of Python security code data and React 18.2.0 code for the security and React models respectively. .

4.2.2 React Dataset

- Base Model Perplexity: 2.34
- Fine-Tuned Model Perplexity: 1.28

4.2.3 Security Dataset

- Base Model Perplexity: 4.16
- Fine-Tuned Model Perplexity: 2.20

4.3 Performance Benchmarking

- Benchmark: Tokens per Second

- Description: The number of tokens per second the model can generate. The higher the number, the better the model. The Benchmarking is done on the same hardware, prompt and using the same hardware for the base model and the fine-tuned model, a Tesla V100.

4.3.1 Inference Speed

- Base Model speed: 80 tokens/second
- Fine-Tuned Model speed: 68 tokens/second
- GPT-4 speed: 45 tokens/second

4.3.2 Security Dataset

- Base Model Accuracy: [Accuracy Value]
- Fine-Tuned Model Accuracy: [Accuracy Value]