Project Proposal: CodeSmile

Recupito Roberto, Palomba Fabio
IGES, Software Engineering for Artificial Intelligence
d.mazza@unisa.it, n.delogu@unisa.it

Project Scope

ML-specific code smells refer to sub-optimal implementation solutions applied within machine learning (ML) pipelines. These code smells may negatively impact the quality and maintainability of ML-enabled systems. Previous studies have shown that these types of smells, if not addressed, can reduce the overall effectiveness of ML systems, making them harder to scale, maintain, and optimize. In this context, CodeSmile is designed to address the detection of these ML-specific code smells. It is a tool that focuses on identifying various patterns of sub-optimal implementation commonly found in ML codebases. By analyzing these patterns, CodeSmile aims to improve the maintainability and quality of ML code by providing insights into problematic code sections that could lead to inefficiencies or difficulties in scaling. Although CodeSmile provides a mechanism for detecting these smells through static analysis, this project proposes an alternative approach based on LLMs and evaluates its feasibility.

Project Ideas

The primary objective of this project is to enhance CodeSmile by developing an LLM-based detection module using the Qwen Coder series. These models, developed independently by Alibaba Cloud, will be utilized for two key tasks: first, artificially injecting code smells into code snippets to generate a diverse training dataset; and second, detecting instances of these code smells following a fine-tuning phase. The decision to leverage the Qwen Coder models stems from their strong and comprehensive coding capabilities, which are comparable to those of GPT-4.

Starting Assets

- CodeSmile Repository: https://github.com/xDaryamo/smell_ai
- LLM frameworks and libraries (e.g., Qwen API, Ollama, scikit-learn).
- CodeSmile Scientific Paper and ML-specific Code Smells definition available at https://arxiv.org/abs/2403.08311

Project Main Steps

1. **Understanding CodeSmile:**
   The first step is to familiarize with the current architecture of CodeSmile, focusing on how the tool detects ML-specific code smells through static analysis. This involves a thorough review of the existing documentation and source code to ensure a solid foundation for integrating the new LLM-based detection module.
2. **Code Smell Injection and Dataset Design:**
   The next step is to design the dataset by artificially injecting ML-specific code smells into real code snippets using the QwenCoder2.5B. This process includes a phase of prompt engineering, where the prompts are carefully designed to guide the models in generating meaningful code smells. These injected smells, along with real instances, will serve as the basis for training the detection model.
3. **Model Training and Fine-tuning:**
   The generated dataset will be preprocessed and structured to ensure it is diverse and representative for the detection task. The QwenCoder2.5B-Instruct models will be trained and fine-tuned to recognize and classify the injected ML-specific code smells. The training process will involve selecting the appropriate configurations to improve their performances.
4. **Validation and Performance Evaluation:**
   Once the models are trained, their performance will be evaluated against a validation dataset. This process involves assessing the models' effectiveness in terms of metrics such as accuracy, precision, recall and F1-score. This step ensures that the most effective model is selected for deployment based on its performances.
5. **Deployement:**
   In the final step, the trained ML module will be integrated into the existing architecture. This will require adapting the detection system, utilized by the WebApp module, to accommodate the new feature.

Minimum Requirements

- The project should provide a workable and well-defined pipeline.
- The project should provide a valid and usable synthetic dataset.
- The module should detect single instances of code smells in a given code snippet (single-labeled detection).

Award Criteria

- Enable the injector to generate code snippets that contain multiple smell instances.
- Enable the model to detect multiple code smells within the same code snippet (multi-labeled detection)