SE4AI project report: CodeSmile

Delogu Nicolò, Mazza Dario (Non frequenta SE4AI)
n.delogu@studenti.unisa.it, d.mazza6@studenti.unisa.it
University of Salerno, Fisciano, Italy

ABSTRACT

CodeSmile is a static analysis tool designed to detect ML-specific code smells: suboptimal implementation patterns that compromise the quality, maintainability, and scalability of machine learning pipelines. Unresolved code smells can degrade system performance, reduce model interpretability, and introduce hidden technical debt. This project explores the feasibility of enhancing CodeSmile through large language models (LLMs). Specifically, the Qwen Coder series is leveraged for two core tasks: (i) generating high-quality synthetic datasets by injecting ML-specific code smells into clean code snippets using chain-of-thought (CoT) prompts and (ii) fine-tuning the model for smell detection. By addressing challenges related to dataset construction, generalization, and classification performance, this study evaluates whether LLMs can complement or enhance traditional static analysis techniques. Preliminary results demonstrate that while synthetic code smell injection can generate diverse datasets that maintain structural correctness, training on such datasets alone limits model generalization to real-world instances. Furthermore, hardware constraints hindered the full exploration of multi-labeled detection tasks, indicating the need for improved computational resources. Nonetheless, the study highlights the potential of LLMs in detecting ML-specific code smells, offering a promising direction for future research.

All materials, including tool documentation, datasets and code are available at https://github.com/xDaryamo/smell_ai.

ACM Reference Format:
Delogu Nicolò, Mazza Dario (Non frequenta SE4AI), n.delogu@studenti.unisa.it, d.mazza6@studenti.unisa.it, University of Salerno, Fisciano, Italy. 2025. SE4AI project report: CodeSmile. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 CONTEXT OF THE PROJECT

Machine learning (ML) has become a cornerstone of modern software engineering, enabling innovation across various industries. However, the rapid growth and deployment of ML pipelines present unique challenges related to code quality and maintainability. Unlike traditional software systems, ML pipelines are highly dynamic, encompassing complex data pre-processing, model training, and evaluation workflows. This complexity gives rise to ML-specific code smells: patterns in the code that deviate from best practices, leading to suboptimal implementations. Such smells can negatively affect scalability, maintainability and efficiency n ML-enabled systems. CodeSmile, a tool for the static analysis of ML codebases, was developed by Recuptio et al. [1] to address these challenges. By identifying ML-specific code smells, it empowers developers to make informed decisions to improve code quality. In this context, the proposed project aims to extend CodeSmile by incorporating a large language model (LLM)-based detection module. This integration leverages the capabilities of the Qwen Coder series, which rival state-of-the-art systems like GPT-4. By employing LLMs, the project seeks to provide a robust and reliable alternative to traditional static analysis methods. Furthermore, the project introduces a tailored dataset generation process, where LLMs are employed to artificially inject code smells into real-world code snippets, creating a training foundation.

2 GOALS OF THE PROJECT

The primary objectives of this project are: (i) to extend CodeSmile by integrating LLM-based detection and injection modules tailored for ML-specific code smells, (ii) to generate a valid synthetic dataset by leveraging LLMs for artificial code smell injection, and (iii) to assess the effectiveness of the detection module in identifying both individual and multiple coexisting code smells.

Following the GQM (Goal-Question-Metrics) paradigm, we provide a structured outline of the goals and associated specifics:

Extending CodeSmile by incorporating an LLM-based detection and injector module for the purpose of broadening its ability to detect, classify, and evaluate ML-specific code smells while also generating high-quality datasets. The assessment is carried out from the viewpoint of Software Engineering Master Degree students, in the context of software engineers and researchers working with ML-enabled systems.

- **Object of study:** The extended tool, featuring both an LLM-based detection module and an injector module for synthetic dataset generation.
- **Purpose:** To extend CodeSmile's functionality by introducing an LLM-based detection module and an injector for generating synthetic datasets with realistic code smell instances. To assess the feasibility of the proposed approach.
- **Quality Focus:** Evaluating the detection module's classification performance (precision, recall, and F1-score) and the realism of the injected code smells.
- **Perspective:** Assessing CodeSmile's performance from the perspective of end users (software engineers and researchers), focusing on detection performances and dataset validity.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Delogu Nicolò, Mazza Dario (Non frequenta SE4AI)
n.delogu@studenti.unisa.it, d.mazza6@studenti.unisa.it
University of Salerno, Fisciano, Italy

- **Context, Environment and Constraints:**
  - **Subjects:** Users include software engineers and researchers developing ML systems.
  - **Objects:** ML projects and code snippets with sub-optimal implementation patterns (may contain more than one code smell).
  - There is limited work on LLM-based detection of ML-specific code smells and dataset generation, making this project exploratory in nature.

To achieve this goal, we arranged our paper around the following research questions (**RQ#**):

RQ1. To what extent can LLMs accurately generate realistic training data with artificially injected code smells?

RQ2. How effective is the new module in detecting and classifying ML-specific code smells?

This research question can be further divided into two sub-questions:

RQ2a. How effective is the module at identifying individual code smells within a single code snippet?

RQ2b. How effective is the module at detecting multiple code smells coexisting within a single code snippet?

With the first research question (RQ1), we aim to evaluate the ability of the Qwen Coder models to generate synthetic datasets that closely mimic real-world ML-specific smells. Since the scarcity of labeled data is a critical bottleneck in this domain, the correctness and diversity of the injected code smells will be essential metrics. If necessary, we will adapt and refine the injection process to address deficiencies, ensuring that the resulting dataset provides a foundation for training and evaluating the detection module. Regarding the second research question (RQ2), we intend to evaluate the effectiveness of the enhanced detection module in identifying and classifying ML-specific code smells. This evaluation will be conducted across two scenarios: individual code smells (RQ2a) and cases with multiple coexisting smells (RQ2b). The performance of the module will be measured using metrics such as accuracy, precision, recall and F1-score to ensure a comprehensive assessment.

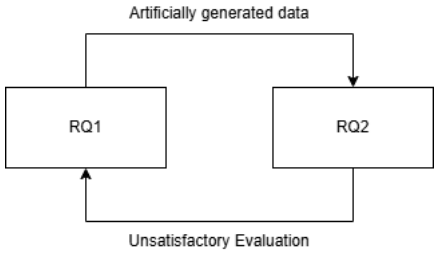

Figure 1: Research Question # input/output diagram

The interdependence between RQ1 and RQ2 is critical; the quality of the datasets generated in RQ1 will directly impact the detection accuracy in RQ2.

3 METHODICAL STEPS TO CONDUCT TO ADDRESS THE GOALS

To achieve the project's objectives, a structured methodology is implemented, covering dataset generation, model training, and performance evaluation.

The first step involves constructing a dataset suitable for training and evaluating the enhanced CodeSmile tool. ML-related code snippets are collected from public repositories and analyzed to identify existing code smells. The dataset is then augmented by injecting ML-specific code smells into clean code snippets through the Qwen Coder 14B model. Chain-of-Thought (CoT) prompts will guide the generation process to ensure that the introduced smells are coherent and contextually appropriate. Once generated, the dataset will undergo validation, where a dedicated evaluation class will assess the syntactic correctness of the snippets and computes similarity between dataset entries (we apply TF-IDF to transform code blocks into numerical vectors based on word frequency and then we compute cosine similarity between blocks).

The second phase focuses on developing and fine-tuning the detection module. The Qwen Coder models are fine-tuned on the augmented dataset. Techniques such as LoRA fine-tuning, early stopping and lowering model complexity (depending on the size of the dataset) will be employed to reduce over-fitting risks.

To evaluate the effectiveness of the proposed approach, the two key research questions are addressed. For RQ1, correctness and similarity metrics will assess the quality of the injected code smells, verifying that synthetic instances closely resemble real-world cases. For both RQ2a and RQ2b, the detection performance will be tested under two distinct conditions. In the first scenario, a 70-30 split is applied to a balanced dataset containing both real and synthetic instances. In the second scenario, the model is trained exclusively on synthetic code smells, while validation is performed using only real-world code smells to assess generalization capabilities. Performance metrics such as accuracy, precision, recall, and F1-score provide insights into the model's ability to correctly classify and detect ML-specific code smells.

4 MODEL DEVELOPMENT

This section outlines the development plan of the LLM-based detection module for CodeSmile. The goal is to create models capable of detecting ML-specific code smells with acceptable performances. Figure 2 illustrates the overall pipeline for model development and deployment.

Data Preparation: In order to train and evaluate the LLM-based detection module, we will generate a dataset using the injector module. The injector will leverage the 14B variant of Qwen, chosen for its broad context size, along with chain-of-thought (CoT) prompts to introduce artificial code smells into clean code snippets. The resulting synthetic dataset will be balanced, containing an equal number of clean and smelly functions. The smelly functions will be specifically tailored to represent each code smell class equally.

Dependent Variables Selection: The detection module will be designed to classify code snippets based on the presence of ML-specific code smells. The dependent variable will be defined as a unary label, indicating the specific code smell detected, if present. Each code

snippet in the dataset will be associated with a label corresponding to one of the code smell classes (or labeled as "no smell" if no smell is detected). Additionally, the detection module will also support a multi-label classification scenario, where a single code snippet may exhibit multiple code smells simultaneously, and thus be assigned multiple labels accordingly.

Independent Variable Selection: The independent variables are the CoT prompts containing code snippets from the balanced dataset (produced by the injector). These prompts include both clean code and smelly code snippets, which have been artificially modified using the injector module. The model will then be exposed to a variety of code smells and learn to identify associated patterns.

Model Fine-Tuning: The models will be fine-tuned using the artificial dataset. Fine-tuning involves optimizing the models on labeled snippets with the goal of minimizing loss and maximizing classification accuracy. To minimize over-fitting, we will employ techniques such as LORA fine-tuning, early stopping during training, and adjusting model complexity depending on the dataset size.

Evaluation: The performance of each model will be evaluated using standard classification metrics: precision, recall, and F1-score. These metrics allow us to assess the models' ability to correctly identify and classify various ML-specific code smells, while minimizing false positives and false negatives.

Validation: The validation process will evaluate the generalization ability of Qwen2.5-Coder-Instruct variants using two distinct experimental setups. Firstly, a balanced dataset evaluation is conducted by applying a 70-30 train-test split, incorporating both real and synthetic instances to ensure a comprehensive assessment. Secondly, a synthetic-to-real validation scenario will be tested, where the models are trained exclusively on synthetic code smells and evaluated on real-world instances.

Deployment: Once the best-performing model is selected, the detection module will be deployed in a dedicated web application via a FastAPI-based service. The web app will allows users to submit code snippets or project folders and receive feedback on potential ML-specific code smells.

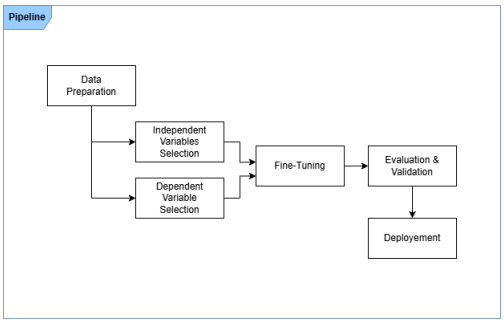

Figure 2: Pipeline Steps

5 PRELIMINARY RESULTS AND FINDINGS

This section presents the preliminary results obtained in response to the research questions RQ1, RQ2a and RQ2b. The findings assess both the effectiveness of the dataset generation process and the performance of the detection model.

5.1 RQ1

To address RQ1, an automated system was developed to generate ML-specific code smells using Large Language Models (LLMs). The integration of Qwen2.5-Coder-14B enabled the efficient creation of code variations embedding different types of code smells while operating within the project's hardware constraints. To ensure dataset validity, two key evaluations were performed. The first involved syntactic validation, which confirmed that 97.38% of all generated code snippets were free from syntax errors. The second evaluation assessed the diversity and uniqueness of the generated snippets by computing similarity measures across dataset entries. Cosine similarity was used to analyze the dispersion of the generated data, leading to the following statistical summary:

- Mean Similarity: 0.0410
- Median Similarity: 0.0284
- Standard Deviation: 0.0501
- Maximum Similarity: 1.0
- Minimum Similarity: 0.0

To provide a visual representation of these findings, a plot illustrating the distribution of similarity values was generated.

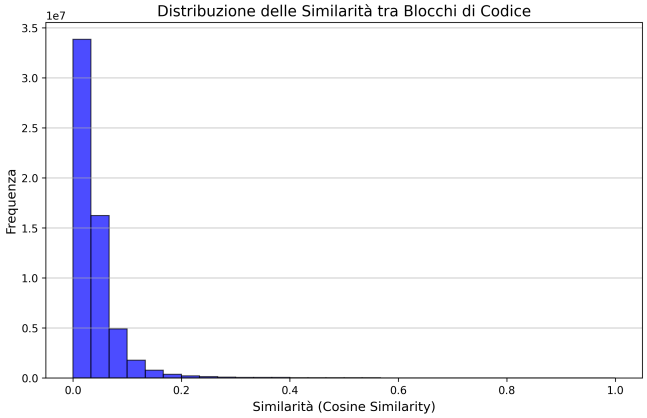

Figure 3: Similarity Distribution

5.2 RQ2a

In order to evaluate the detection models' performance, experiments were conducted using different configurations of Qwen2.5-Coder-Instruct, specifically the 0.5B, 1.5B, and 3B parameters variants. The 3B model was found to be the most effective, achieving the highest performance. Two different evaluation scenarios were considered:

- **Balanced dataset evaluation**: A 70-30 train-test split was applied, using both real and synthetic instances to ensure a comprehensive assessment.
- **Synthetic instances against real instances**: The model was trained exclusively on synthetic code smells, while validation was performed using only real-world smells to measure its ability to generalize beyond artificially generated examples.

Delogu Nicolò, Mazza Dario (Non frequenta SE4AI)
n.delogu@studenti.unisa.it, d.mazza6@studenti.unisa.it
University of Salerno, Fisciano, Italy

The classification results for the best-performing model (3B) is summarized in the following tables:

Classe	Precision	Recall	F1-score
Unnecessary Iteration	0.80	0.93	0.86
Hyperparameter Not Explicitly Set	0.84	0.88	0.86
NaN Equivalence Comparison Misused	0.98	0.97	0.98
Empty Column Misinitialization	0.98	0.96	0.97
Gradients Not Cleared Before Backward Propagation	0.82	0.87	0.85
TensorArray Not Used	0.94	0.93	0.94
Dataframe Conversion API Misused	0.90	0.88	0.89
Deterministic Algorithm Option Not Used	0.99	0.99	0.99
Chain Indexing	0.88	0.76	0.81
No Smell	0.98	0.98	0.98
Matrix Multiplication API Misused	0.97	0.91	0.94
Merge API Parameter Not Explicitly Set	0.93	0.93	0.93
Memory Not Freed	0.93	0.90	0.92
Broadcasting Feature Not Used	0.98	0.97	0.98
In-Place APIs Misused	0.83	0.74	0.78
Columns and DataType Not Explicitly Set	0.89	0.94	0.92
PyTorch Call Method Misused	0.94	0.92	0.93
Micro avg	0.95	0.95	0.95
Macro avg	0.92	0.91	0.91
Weighted avg	0.95	0.95	0.95
Samples avg	0.95	0.95	0.95

Table 1: Balanced Split Scenario

Class	Precision	Recall	F1-score
Deterministic Algorithm Option Not Used	1.00	0.10	0.18
Chain Indexing	0.00	0.00	0.00
Empty Column Misinitialization	0.00	0.00	0.00
Matrix Multiplication API Misused	0.25	0.09	0.12
In-Place APIs Misused	0.00	0.00	0.00
Columns and DataType Not Explicitly Set	0.93	0.09	0.17
Merge API Parameter Not Explicitly Set	0.00	0.00	0.00
Dataframe Conversion API Misused	0.09	0.06	0.07
PyTorch Call Method Misused	0.00	0.00	0.00
NaN Equivalence Comparison Misused	0.00	0.00	0.00
Memory Not Freed	0.00	0.00	0.00
Gradients Not Cleared Before Backward Propagation	0.48	0.08	0.13
Hyperparameter Not Explicitly Set	0.50	0.03	0.06
Unnecessary Iteration	0.00	0.00	0.00
Micro avg	0.47	0.03	0.03
Macro avg	0.27	0.02	0.03
Weighted avg	0.54	0.03	0.04
Samples avg	0.04	0.04	0.04

Table 2: Synthetic Instances against Real Instances

5.3 RQ2b

During the project execution, certain hardware limitations gravely affected the performance of the models for both multiple code smell injection and multi-labeled detection. The high computational requirements of such approaches exceeded the available resources. Since we were unable to overcome this limitation, we found ourselves unable to properly assess RQ2b. Henceforth, our focus shifted toward enhancing prompt engineering to maximize the capabilities of the available models.

6 IMPLICATIONS OF THE RESULTS

6.1 RQ1

The results obtained highlight a low average similarity among generated snippets, suggesting a high level of variability in the synthetic dataset. While a maximum similarity value of 1.0 indicates that some instances may be almost identical, these cases are isolated and do not significantly impact dataset diversity. Moreover, they indicate that the injection process maintained the structural correctness of the code, minimizing the need for corrections.

6.2 RQ2a

Balanced Split Scenario. The model achieved an overall accuracy of 94.46%, demonstrating strong performance across different code smell categories. The macro-averaged F1-score of 0.95 highlights a well-balanced trade-off between precision and recall, confirming the model's ability to detect ML-specific code smells effectively. These results suggest that the model generalizes well when trained on a dataset containing both real and synthetic code smells.

Synthetic instances against real instances. In contrast, when the model was trained exclusively on synthetic data and evaluated using real-world instances, its performance dropped significantly, with a macro-averaged F1-score of only 0.03. Several classes, including Memory Not Freed and PyTorch Call Method Misused, had recall values of 0, indicating that the model failed to recognize them. A key factor contributing to this performance gap is the difference in validation dataset size between the two scenarios. The synthetic-to-real evaluation used a significantly smaller validation dataset compared to the balanced split scenario. This discrepancy could have led to increased variability in performance metrics, potentially exaggerating the observed drop in effectiveness. Additionally, the limited real-world validation set may have contained a distribution of smells that the model was not adequately exposed to during training, further hindering its ability to generalize.

6.3 RQ2b

The inability to fully assess RQ2b due to hardware constraints limits the scope of our findings in the context of multi-labeled code smell detection. This limitation primarily stems from the high computational requirements needed for injecting multiple code smells into a single snippet and the subsequent multi-label classification tasks. The existing hardware infrastructure was unable to handle these requirements within the allocated resources. However, this challenge is not a reflection of the model's capability but rather a matter of computational capacity. With access to more powerful hardware, it is expected that larger context window variants of Qwen Coder could perform multi-labeled detection more efficiently. Therefore, future research should focus on improving computational resources or utilizing more efficient techniques for handling multiple code smells in a single code snippet.

7 CONCLUSION

This study explores the integration of Large Language Models (LLMs) for detecting ML-specific code smells, with a focus on evaluating the performance of models trained on synthetic datasets and their generalization to real-world scenarios. The findings demonstrate that LLM-generated synthetic code smells exhibit a high level of variability, which, along with the structural correctness of the generated code, makes them suitable for use in model training. The results from the balanced split scenario, where the model achieved an overall accuracy of 94.46% and a macro-averaged F1-score of 0.95, suggest that the model is highly effective at detecting ML-specific code smells when trained on both real and synthetic data. However, when the model was exclusively trained on synthetic data and evaluated on real-world instances, its performance significantly dropped. The gap in performance can be attributed to the smaller validation set and potential distributional differences between synthetic and real-world smells. This reinforces the need for a more balanced training approach, combining both synthetic and real instances to improve the model's robustness and generalization capabilities. The study also encountered hardware limitations in assessing multi-labeled code smell detection, which prevented the full exploration of this area. The high computational requirements for injecting multiple code smells and handling multi-label classification tasks highlighted the need for more powerful hardware to address these challenges effectively.

To sum up, while the current model shows potential for detecting ML-specific code smells, it also underscores the importance of using robust computational resources. Future research should explore methods to enhance dataset diversity, improve hardware infrastructure, and further refine multi-labeled detection capabilities. These efforts will ultimately contribute improve the maintainability and quality of ML-enabled systems, ultimately benefiting the broader machine learning community.

REFERENCES

[1] Gilberto Recupito, Giammaria Giordano, Filomena Ferrucci, Dario Di Nucci, and Fabio Palomba. 2024. When Code Smells Meet ML: On the Lifecycle of ML-specific Code Smells in ML-enabled Systems. *arXiv e-prints*, Article arXiv:2403.08311 (March 2024), arXiv:2403.08311 pages. https://doi.org/10.48550/arXiv.2403.08311 arXiv:2403.08311 [cs.SE]