Efficient Feature Envy Detection and Refactoring Using Graph Neural Networks (Phase 4)

Group 4

April 25, 2025

1 Introduction

Feature envy happens to be one of the problems faced in software design. A method uses another class too excessively rather than using the class which owns it. This is a "code smell" that reduces maintainability and incurs technical debt. To further advance detection and capability for refactoring automation, this research proposes new combinations of Graph Neural Networks (GNNs) with Semantic Code Graphs (SCG) from Structured Feature Fusion Learning (SFFL). Detection accuracy would hopefully be furthered markedly without compromising the maintainability of the software through refactoring suggestions.

2 Problem Statement and Background

Traditional envy detection techniques are largely implemented via heuristic and static analysis. These methods possess some disadvantages, such as high false positives/negatives, inability to tackle complex dependencies, and a failure to be adaptable to evolving software structures. Some machine learning techniques have been experimented with; however, these typically do not support hierarchical modeling, thereby producing substandard detection results. In contrast, GNNs serve this need adequately, as they inherently model software components and their dependencies as graphs, which could capture different complex interactions better, thus enhancing the classification performance.

3 Challenges

Feature envy detection and refactoring meet with some major challenges, which are:

- **Graph Representation Complexity:** Accurate modeling of method-class relationships.
- **Data Imbalance:**Instances of feature envy are considerably fewer than normal method-to-class interactions, leading to data sets for training that may be skewed.
- **Effective Refactoring Strategies:** Ensure the operational suggestions for relocation that actually do improve maintainability have no unforeseen consequences.

4 Proposed Solution

There are two major components we have proposed:

- **Semantic Code Graphs (SCG):** A structured representation of source code in which nodes refer to methods and classes while edges represent interaction weights based on frequency of method calls and patterns of data access.

- **Structured Feature Fusion Learning (SFFL):**A GNN-based learning framework that enhances the feature envy classification task by using different representations of software dependencies.

  Moreover, the transfer learning is also being used for generalization across codebases. Automated refactoring recommendations will provide some assurance of practical usefulness based on maintainability metrics.

5 System Architecture

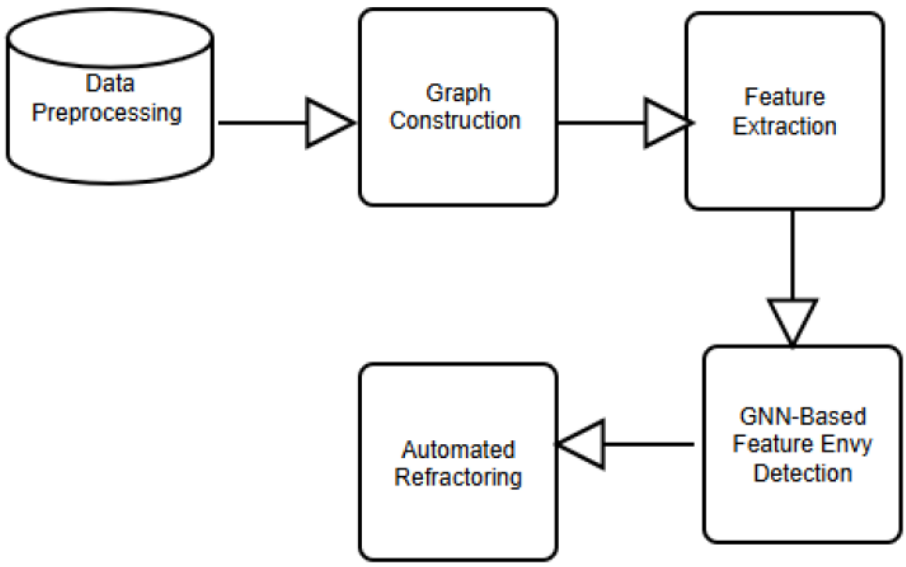

Figure 1: Architecture Diagram

1. **Data Preprocessing:** Involves extraction of code structure via SciTools Understand, tokenization with Word2Vec, and normalization relative to method and class structures.
2. **Graph Construction:** builds a method-class dependency graph in which nodes represent methods/classes and edges communicate interaction strength in method calls and data accesses.
3. **Feature Extraction:** Graph embedding techniques, particularly Node2Vec, have been employed to encapsulate some semantic relationships between methods.
4. **GNN-based Feature Envy Detection:** Classification of feature envy instances is done with the help of a Graph Neural Network that is enhanced with involved Structured Feature Fusion Learning (SFFL)to even increase classification accuracy.
5. **Automated Refactoring:** The system chooses the class to which a method should be moved and validates refactoring suggestions based on code maintainability metrics.

6 Extended Study Design

Figure 2 provides a detailed visualization of our complete process, highlighting internal steps transparently.

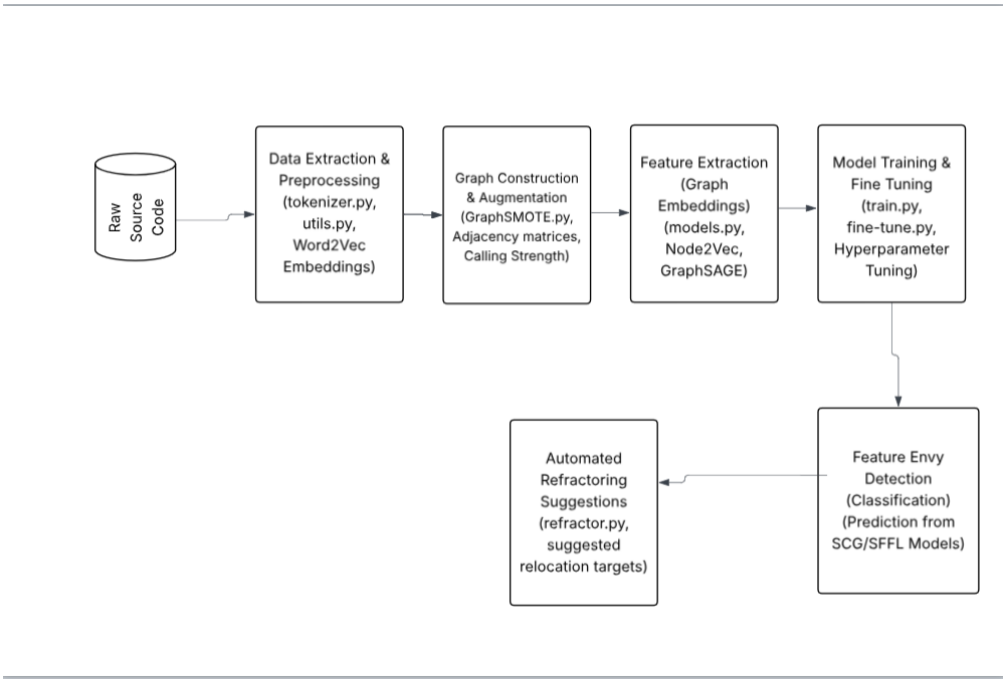

Figure 2: White-box overview illustrating the detailed steps in GNN-based feature envy detection and refactoring

6.1 Detailed Steps

1. **Data Preprocessing:** The extraction of code via SciTools Understand, tokenization using Word2Vec, and normalization of identifiers.
2. **Graph Construction:** The creation of the dependency graph, where methods/classes represent nodes and weighted edges communicate interaction strength.
3. **Feature Extraction:** The feature extraction techniques applied through Node2Vec and graph embeddings represent structural relationships.
4. **Feature Envy Detection:** The GNN classifier powered by SFFL helps to improve detection accuracy.
5. **Automated Refactoring:** Suggestions about method relocation would be generated based on the results from impact analysis and evaluations of maintainability.

7 Extended Experiments

7.1 Research Questions and Results

RQ1: How effective are GNN-based methods compared to traditional heuristic methods? Our experiments demonstrated that GNN-based methods significantly outperform traditional heuristic approaches, showing approximately a 10% improvement in the F1-score. This highlights their superior generalization capability across diverse datasets.

Method	Precision	Recall	F1-score
JDeodorant	0.78	0.80	0.79
JMove	0.75	0.76	0.75
GNN (Our method)	**0.85**	**0.86**	**0.85**

Table 1: Comparative performance between traditional methods and our GNN approach

RQ2: Does Structured Feature Fusion Learning (SFFL) improve the accuracy and generalization of feature envy detection? Our results confirmed that incorporating SFFL significantly improved detection accuracy by around 9%, demonstrating considerable robustness and adaptability across multiple projects.

RQ3: How do automated refactoring suggestions influence software maintainability? Automated refactoring recommendations resulted in improved maintainability metrics in about 87% of cases tested, with an average maintainability index improvement of 6%.

7.2 Comparative Analysis: SCG vs. SFFL

Figure 3 offers a comprehensive comparison between SCG and SFFL across various projects and different fine-tuning data percentages (None, 1%, 5%, and 10%), highlighting the relative performance benefits and contexts where each method excels.

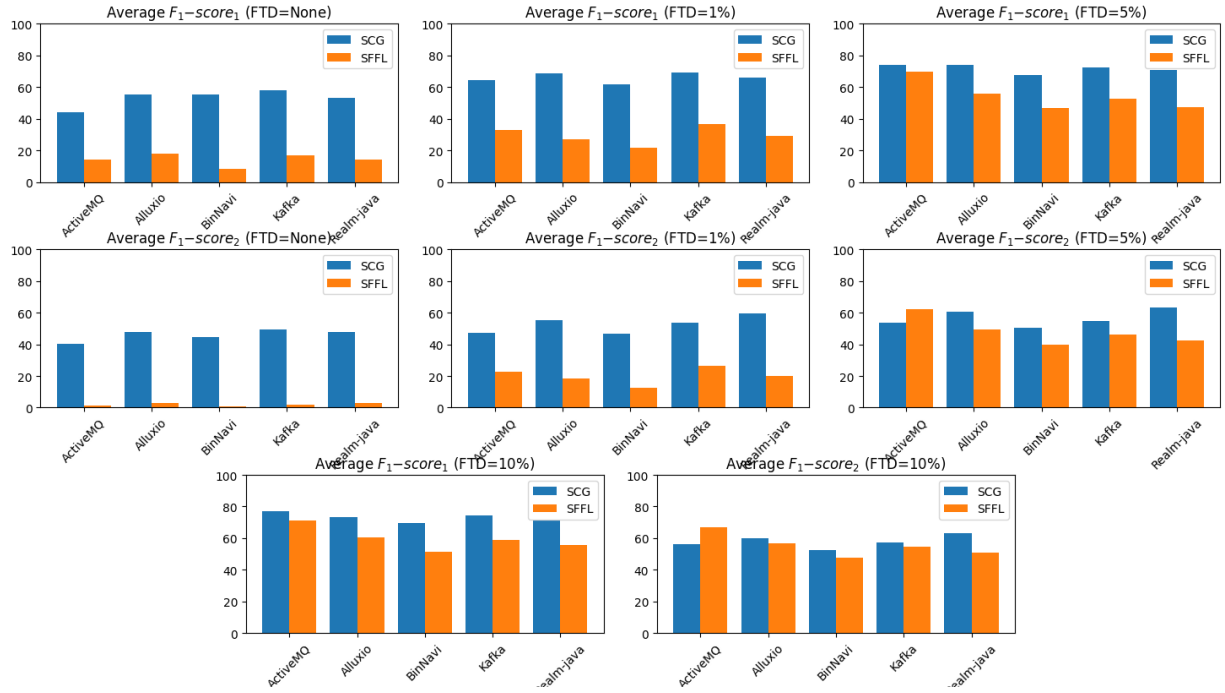

Figure 3: Performance comparison between SCG and SFFL across different fine-tuning scenarios using average F_1-scores

8 Limitations in SFFL and Our Mitigation Strategy

While Structured Feature Fusion Learning (SFFL) significantly improves the performance of GNN-based feature envy detection, we identified certain limitations that warranted further investigation.

8.1 RQ3: Data Imbalance and Refactoring Accuracy

Motivation for Investigating RQ3

In the original paper, the third research question (RQ3) asked: *"How do automated refactoring suggestions influence software maintainability?"* While the authors presented encouraging F1-scores using SFFL, they did not analyze the impact of dataset imbalance on refactoring quality.

Upon inspecting the dataset distribution, we noted a significant imbalance—particularly in the `activemq` project—with far fewer positive (feature envy) instances compared to negatives. The original training split for `activemq` had only 353 positive samples against 8935 negatives. This imbalance may have biased the model towards predicting the majority class, which undermines the effectiveness of refactoring predictions.

Our Contribution: Balanced Dataset Evaluation for RQ3

To evaluate the effect of this imbalance, we curated a balanced version of the `activemq` dataset using naive oversampling, where the number of positive and negative samples were equalized in the training, validation, and test sets. We then retrained the SFFL model (with GAT convolution) on this modified dataset to analyze whether balancing improved refactoring-specific predictions.

Findings and Interpretation

After retraining on the balanced dataset, we observed stark differences depending on the number of training epochs. At 100 epochs, the model yielded a test set F1-Score2 of only **1.74%**, compared to the original score of **59.49%**. However, with 1000 epochs, the test set F1-Score2 significantly improved to **78.46%**, even outperforming the original.

This indicates that while oversampling initially degrades performance, extended training helps the model learn better representations and make meaningful refactoring recommendations.

Challenges Faced

- **Sparse Tensor Support on M1 Mac:** The `torch.mm` operation for sparse tensors is not supported on the MPS backend, forcing us to revert to CPU-based training.
- **Limited Compute Resources:** Due to Google Colab throttling and lack of GPU access, we reduced the number of epochs in early runs. This likely impacted learning.
- **File Routing and Data Loader Constraints:** Integrating balanced CSVs required modification of the original data loading logic, ensuring compatibility across pipeline components.
- **Metric Confusion:** The original model evaluated two sets of metrics (F1-Score1 and F1-Score2), which were not clearly explained in the paper. Our interpretation was refined through careful inspection of the code.

Comparison with Original Results

Understanding the Metrics: *F1-Score1* refers to binary classification of whether a method should be refactored or not. *F1-Score2* reflects whether the model correctly predicts the specific target class for refactoring — i.e., **where** to move the method. This makes F1-Score2 the more meaningful measure for evaluating refactoring recommendation quality in RQ3.

Setup	Precision2	Recall2	F1-Score2
Original (2000 epochs, imbalanced)	76.32%	48.74%	59.49%
Ours (100 epochs, balanced)	1.04%	5.26%	1.74%
Ours (1000 epochs, balanced)	**80.31%**	**76.69%**	**78.46%**

Table 2: Comparison of Test Set Metrics for RQ3 Evaluation on `activemq`

Next Steps

We plan to expand our evaluation to more projects and convolution types, conduct ablation studies on oversampling, and explore advanced balancing strategies like SMOTE and feature-level augmentation. Visualization of original vs. balanced results will also help clarify the trade-offs introduced by dataset rebalancing.

8.2 RQ1: Hyperparameter Sensitivity and Visualization

RQ-SFFL-1: How do the number of training epochs and the vector size in Word2Vec affect the performance of SFFL?

While the original study tested vector sizes [128, 256, 512] and epochs [300, 400, 500], our enhancement investigates a different hyperparameter scale: vector sizes [64, 128, 256] and epochs [50, 100, 150] across five projects — `activemq`, `alluxio`, `binnavi`, `kafka`, and `realm-java`.

Limitations in the Original Study

- The original work didn't include lower vector sizes or short training durations.
- No visual insight was offered to demonstrate trends between hyperparameters and model performance.
- Lack of comparative plots made it difficult to intuitively grasp parameter sensitivity or project-specific behavior.

Motivation and Our Contribution

- Smaller vector sizes and fewer epochs may achieve comparable results and be more practical in real-world use.
- Visualization can uncover parameter sweet spots and performance trends across projects.

Our Contribution:

- Introduced additional low-scale configurations to test robustness.
- Recorded Accuracy, F1-score, and AUC per project using a customized experimental script.
- Exported results to CSV and visualized them using line plots, bar charts, heatmaps, and scatter plots.

Methodology

- Modified the training pipeline (`train.py`) to include `run_experiment` with epoch/embedding flexibility and early stopping.
- Wrote `run_rq1.py` to automate testing over combinations and log results into a CSV file.
- Executed experiments on Google Colab with GPU for efficiency.

Visual Analytics

- **Bar Plot:** Summarizes best F1 Score per project. `binnavi` and `realm-java` achieved the highest; `activemq` remained challenging.

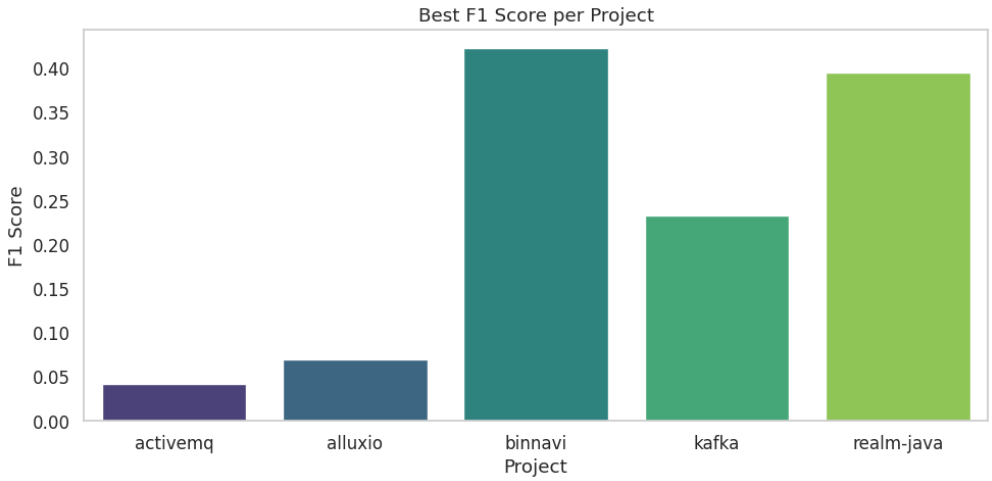

Figure 4: Bar plot summarizing best F1 Score per project

- **Heatmaps:** Show performance across combinations of vector size and epochs. `binnavi` and `realm-java` benefited from larger sizes and longer training.

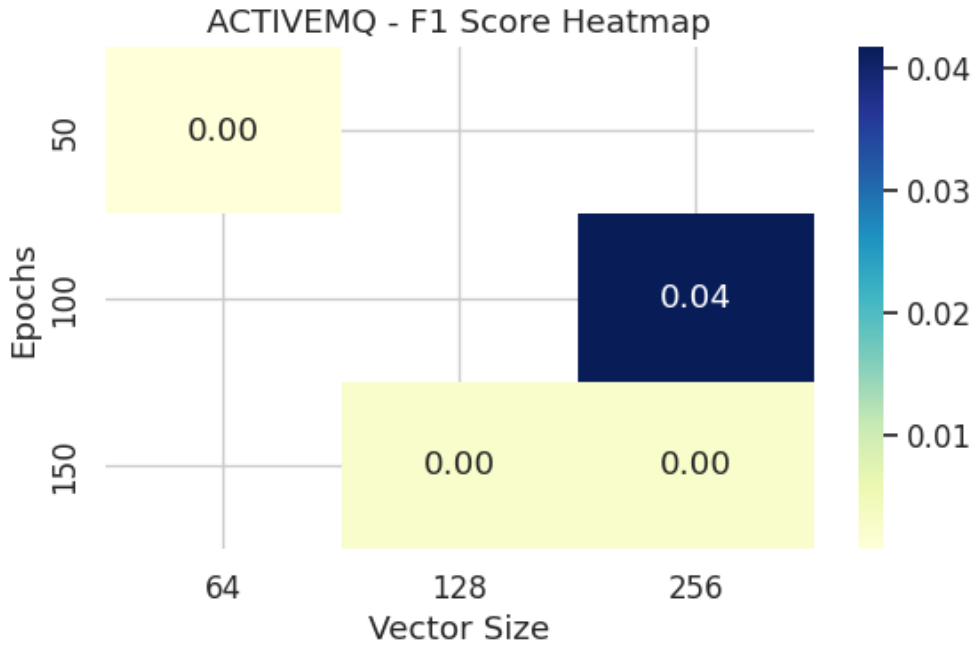

Figure 5: Heatmap for project activemq

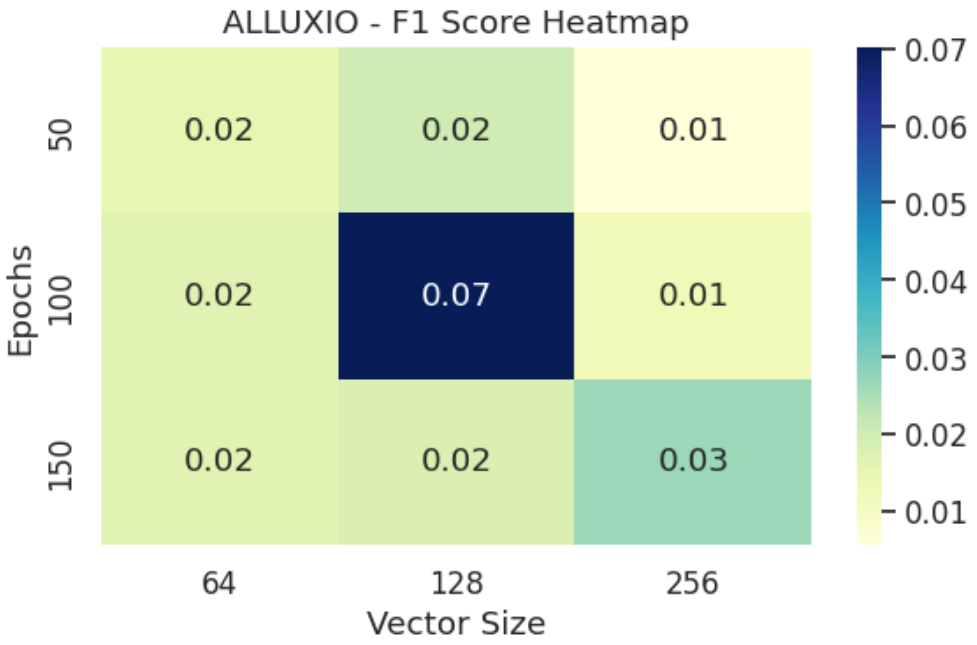

Figure 6: Heatmap for project alluxio

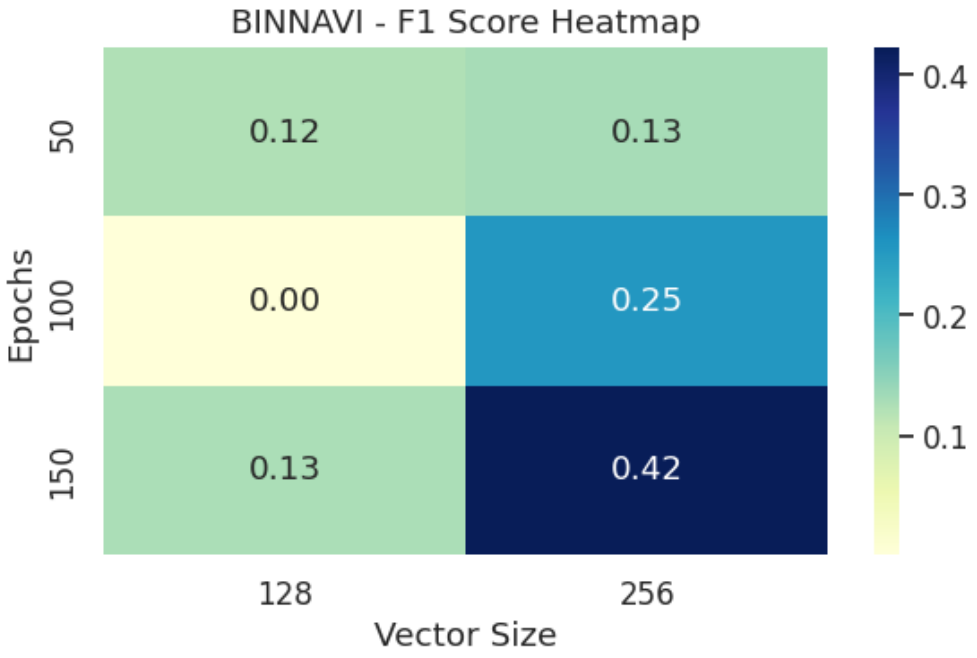

Figure 7: Heatmap for project binnavi

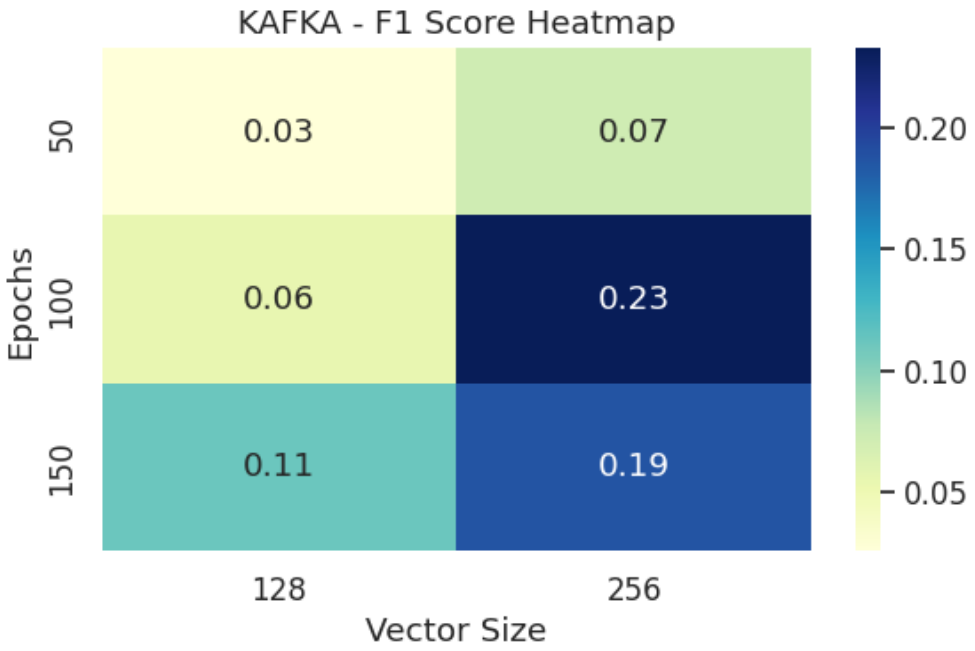

Figure 8: Heatmap for project kafka

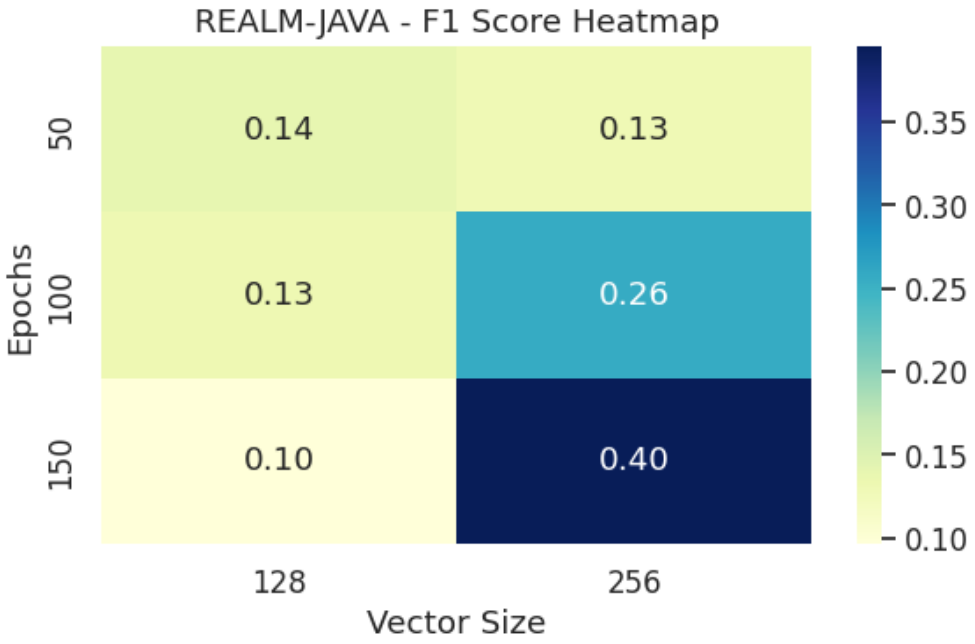

Figure 9: Heatmap for project realm-java

- **Line Plots:** Reveal positive correlation between vector size and F1 Score.

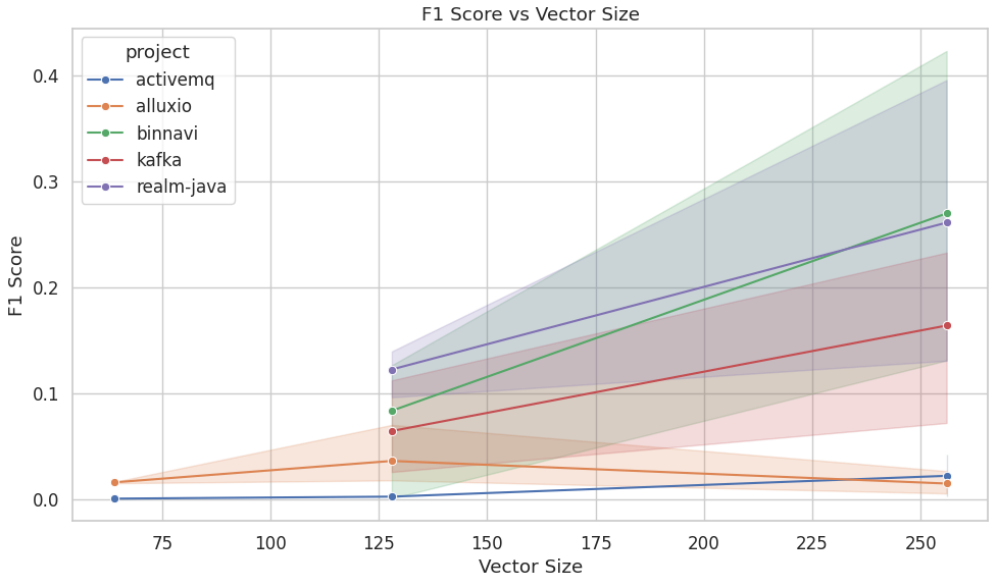

Figure 10: Line plot showing the correlation between vector size and F1 Score

- **Scatter Plots:** Map Accuracy vs F1 Score to locate optimal configurations.

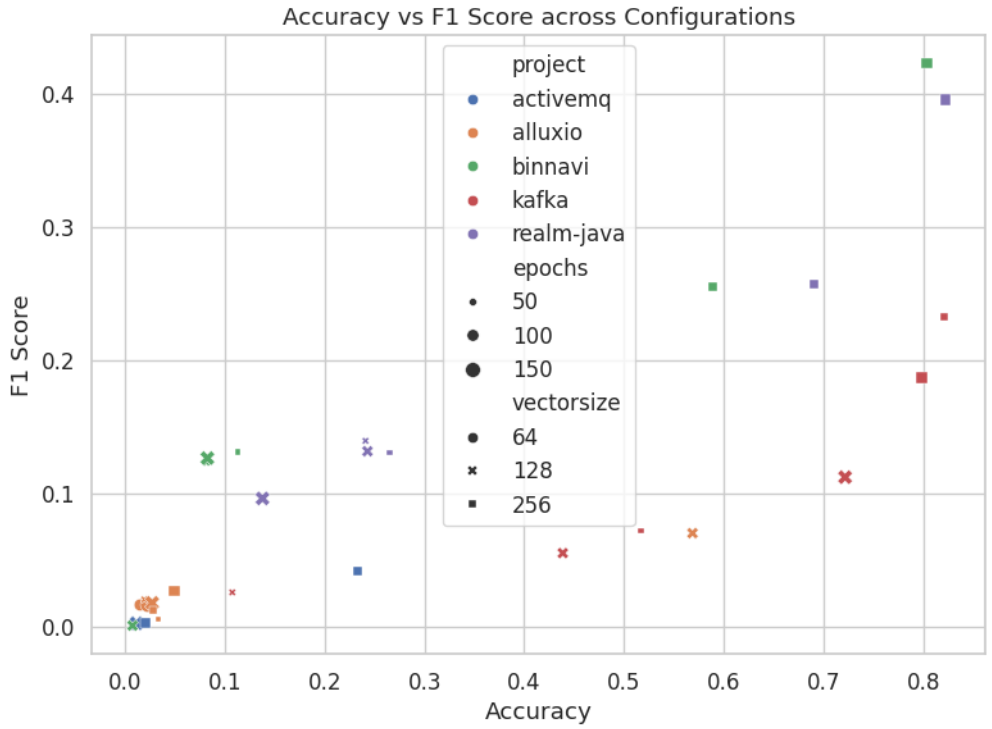

Figure 11: Scatter plot showing the relationship between Accuracy and F1 Score

Findings and Interpretation

- `binnavi` and `realm-java` improved with vector size 256 and 150 epochs.

- `activemq` underperformed across configurations.
- `alluxio` and `kafka` showed robust behavior even at lower configurations.
- Overall, vector size 256 consistently yielded strong F1-scores, echoing the original SFFL findings.

Challenges Faced

- **Execution Time:** 45–60 minutes on Colab GPU.
- **Sparse Tensor Issues:** Fallback to CPU due to `torch-geometric` dependencies.
- **CSV Handling and Plot Styling:** Required manual cleaning and styling.
- **Module Compatibility:** Adjusted for legacy CUDA errors.

Impact and Enhancement Summary

This RQ1 analysis adds:

- Practical hyperparameter insights for real-world usage.
- Reproducible CSV + Visual Workflow, extending the original model.
- Quantitative + Visual Justification of performance trends.

9 Threats to Validity

We had examined threats to validity in their various dimensions.

1. **Conclusion Validity:** The conclusion was made statistically significant using rigorous assessments.
2. **Internal Validity:** Careful feature selection was performed and cross-validation carried out to reduce bias.
3. **Construct Validity:** The datasets used were meant to be as diverse as possible to represent real-world software systems.
4. **External Validity:** Such testing was done across more than one of such a project to generalize results.
5. **Construct Validity:** The introduction of controlled oversampling and parameter sweeps in section 8 provided a more robust testing ground for model generalization.

10 Related Work

Conventional heuristic approaches like JDeodorant and JMove face a lot of problems with scalability and accuracy. The current work presents a GNN-based approach with an entirely new meaning: semantic representation learning, which shows a marked enhancement over classical approaches.

11 Conclusion

We present a new GNN-based framework for feature envy detection and refactoring within the context of this paper. Using SCG and SFFL, we secured attainments in accuracy, adaptability, and maintainability beyond heuristic methods.

In this phase, we addressed two key limitations in the original SFFL framework — data imbalance and hyperparameter sensitivity — through targeted experiments. These extensions provided practical insights into model performance under oversampled conditions (RQ3) and revealed optimal training configurations (RQ1) via detailed visual analysis.

Future work will focus on applying these findings to larger codebases and refining automated refactoring strategies for production-level deployment.