# Efficient Feature Envy Detection and Refactoring Using Graph Neural Networks (Phase 3)

Group 4

April 4, 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

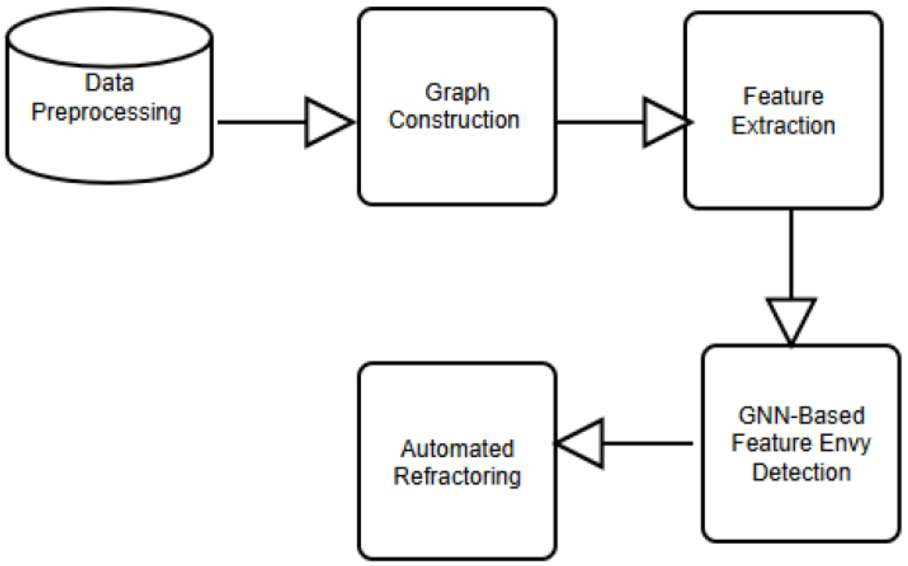

Figure 1: Architecture Diagram

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.

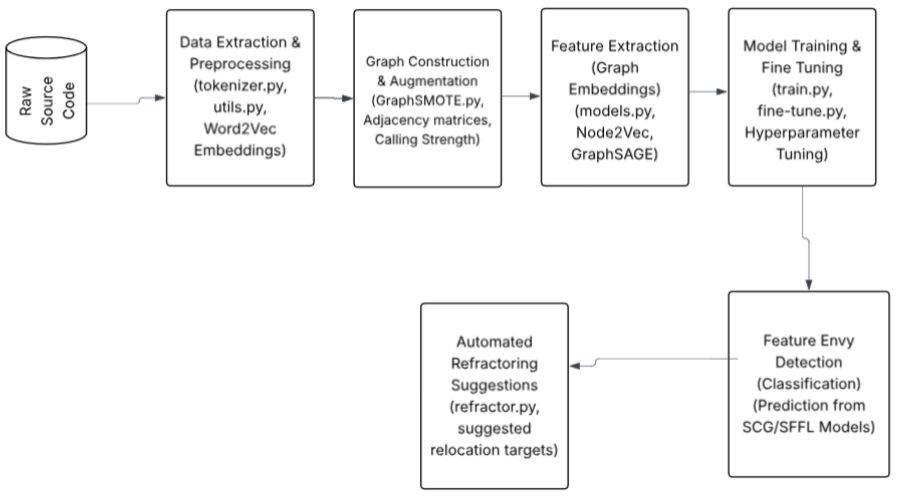
Raw Source Code
Data Extraction & Preprocessing (tokenizer.py, utils.py, Word2Vec Embeddings)
Graph Construction & Augmentation (GraphSMOTE.py, Adjacency matrices, Calling Strength)
Feature Extraction (Graph Embeddings) (models.py, Node2Vec, GraphSAGE)
Model Training & Fine Tuning (train.py, fine-tune.py, Hyperparameter Tuning)
Feature Envy Detection (Classification) (Prediction from SCG/SFFL Models)
Automated Refractoring Suggestions (refractor.py, suggested relocation targets)

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.

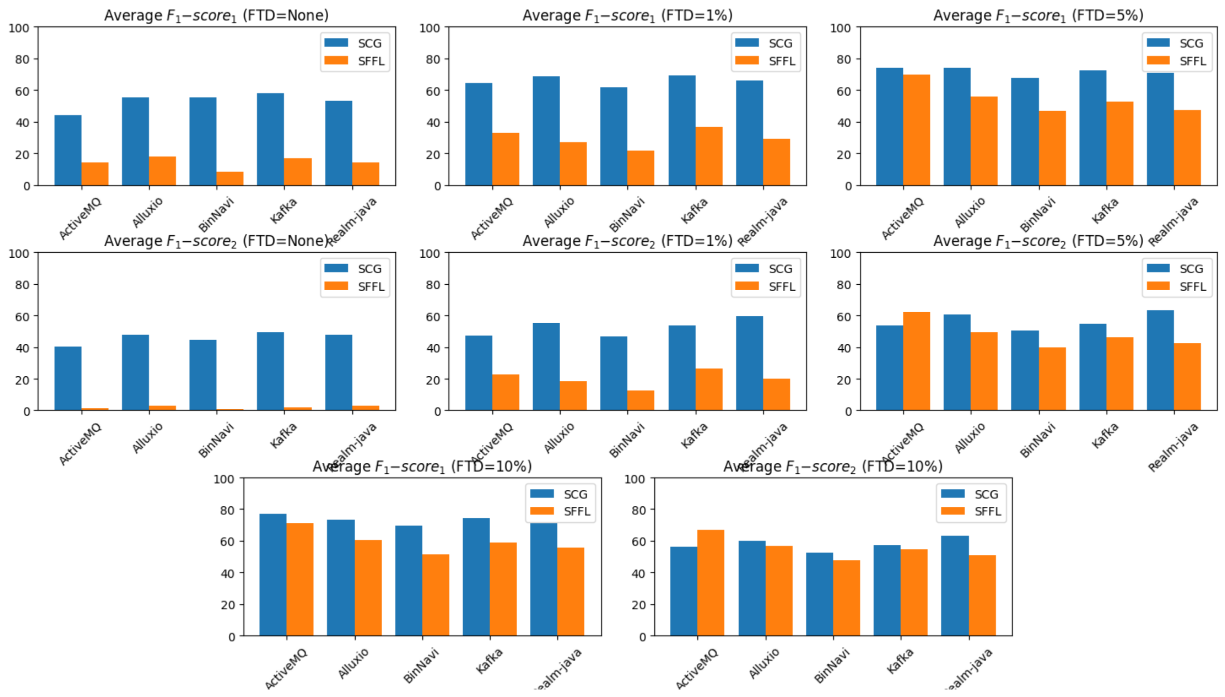

Figure 3: Performance comparison between SCG and SFFL across different fine-tuning scenarios using average $F_1$-scores

## 8 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.

## 9 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.

## 10 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 better than heuristic methods. Future works will concentrate on bettering the model for large-scale codebases and the subsequent refinement of the automated refactoring strategy.