Recommending Move Method Refactoring Opportunities Based on Feature Fusion and Deep Learning

Yang Zhang, Zhenggang Gu, Nan Zhang, and Kun Zheng

***Abstract*—The *Move Method* refactoring is crucial for mitigating the *Feature Envy* code smell, which enhances cohesion and reduces coupling by relocating methods to more suitable classes. Existing deep learning approaches often suffer from redundant features, limiting model generalization. To address this, this paper introduces *GMove*, a novel approach leveraging feature fusion and a hybrid deep learning architecture (Bi-LSTM and CNN branches) to recommend refactoring opportunities. By fusing semantic, structural, and metric features from a constructed 16,828-sample dataset, *GMove* effectively filters redundant information. Experimental results demonstrate that *GMove* achieves a high synthetic F1 score of 97.7% and significantly outperforms state-of-the-art refactoring tools, showing an average F1 improvement of 9.7% over the strongest modern baseline, affirming its effectiveness and novel fusion strategy.**

Link to graphical and video abstracts, and to code:
https://latamt.ieeer9.org/index.php/transactions/article/view/10060

***Index Terms*—Move Method, Refactoring, Feature Envy, Deep learning, Feature Fusion**

I. Introduction

FEATURE ENVY is a recognized code smell where a method of one class shows excessive interest in the data (fields or methods) of another class, leading to high coupling and low cohesion, which severely hinders software maintainability and evolution [1], [2]. To mitigate this, the *Move Method* refactoring—relocating the 'envious' method to its most suitable class—is one of the most common and effective corrective operations [3], [4].

Existing automatic *Move Method* recommendation tools generally fall into two categories: Heuristics-based and Machine Learning/Deep Learning-based [5]–[13]. Heuristics-based approaches use static analysis and software metrics to define rules for identifying 'envious' methods. For instance, JDeodorant [7] applies distance-based metrics, and JMove [6] calculates the similarity of dependency sets to suggest refactoring opportunities. While effective in reducing system coupling, these methods often struggle with complex code semantics and may yield inaccurate recommendations for smaller methods [6], [8], [9].

Machine Learning-based approaches leverage extracted code features to train classifiers. RMove [10] pioneered the use of deep learning by automatically learning structural and semantic representations from code snippets, although its feature concatenation strategy can lead to feature redundancy, impacting model generalization. More recent advancements have pushed the state-of-the-art: HMove [11] utilizes a hypergraph neural network to model complex, multi-entity dependencies, demonstrating strong performance against existing tools. Similarly, Liu et al. [13] and Ma et al. [12] have significantly improved detection accuracy by incorporating real-world data and advanced pretrained language models like CodeT5 [14] to better capture deep semantic relationships.

Despite these advancements, several challenges persist in automatic *Move Method* recommendation. Firstly, a major limitation in deep learning approaches, exemplified by RMove, is feature redundancy, where combining semantic and structural representations leads to duplicated information, ultimately hindering model generalization. Secondly, traditional methods that rely on fixed metrics or heuristic rules often fail to capture the complex and nuanced semantics and structures of code, resulting in inaccurate recommendations. Ultimately, the development of robust deep learning models is hindered by the scarcity of large-scale, high-quality, and non-redundant datasets necessary for effective training in real-world scenarios.

To effectively address these challenges, we introduce *GMove*, a novel approach that recommends *Move Method* refactoring by leveraging advanced feature fusion within a hybrid deep learning architecture. To enable robust training, we first constructed a large-scale synthetic dataset of 16,828 *Move Method* instances extracted from 454 Java projects. GMove systematically extracts and combines three crucial feature types: semantic, structural, and metric features. Critically, to combat feature redundancy and improve generalization, GMove employs a feature fusion model comprising a Bi-directional Long Short-Term Memory (Bi-LSTM) branch and two Convolutional Neural Network (CNN) branches, with the final classification performed by a Fully Connected Neural Network (FCNN). Our evaluations demonstrate the method's superiority, showcasing an average F1 improvement of 9.7% to 65.4% compared to state-of-the-art tools (PathMove, JDeodorant, JMove, RMove, FeTruth, FePM and HMove) on a publicly available dataset, confirming the effectiveness and applicability

The associate editor coordinating the review of this manuscript and approving it for publication was Giner Alor-Hernández (*Corresponding author: Yang Zhang*).

Yang Zhang, Z. Gu, and K. Zheng are with the School of Information Science and Engineering, Hebei University of Science and Technology, Shijiazhuang, Hebei, China (e-mails: zhangyang@hebust.edu.cn, zhengkun@hebust.edu.cn, and 2021114046@stu.hebust.edu.cn).

N. Zhang is with HBlS Digital Technology Co. Ltd, Shijiazhuang, Hebei, China (e-mail: zhangnan02@hbisco.com).

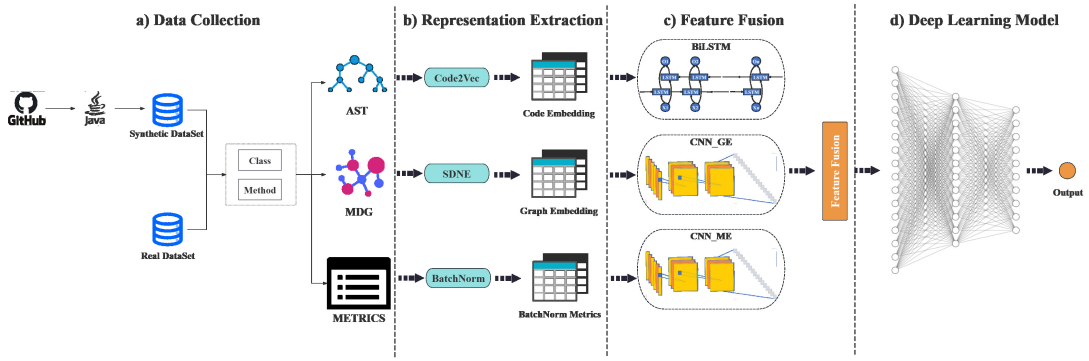

Fig. 1. Overview of the proposed approach GMove including data collection, representation extraction, feature fusion and deep learning model.

of our feature fusion strategy.

The main contributions of this paper can be summarized as follows.

- We propose a novel approach called *GMove* to recommend *Move Method* refactoring opportunities based on deep learning and feature fusion with semantic features, structural features, and metric features.
- We evaluate the effectiveness of *GMove* by addressing six research questions. Our approach demonstrates an increase of 3.2% in recall and 9.7% in F1 compared to state-of-the-art tools, such as PathMove, JDeodorant, JMove, RMove, FeTruth, FePM, and HMove.

II. Motivation

This section presents the motivation behind this research to demonstrate the rationale compelling it forward. How to accurately identify movable methods and their target classes is the primary concern of *Move Method* refactoring. JMove [6] compares the similarity of dependencies between a method and possible target classes to recommend the refactoring opportunity. RMove [10] learns structural and semantic representations from code fragments and trains machine learning classifiers to recommend *Move Method* refactoring. However, these recommendation strategies have several related problems.

RMove's Feature Redundancy Problem: A key limitation of prior deep learning approaches like RMove is their reliance on simple feature concatenation. We hypothesized that merely concatenating structural and semantic embeddings introduces significant feature redundancy, which can impair model performance and generalization.

To validate this, we conducted a preliminary experiment where we replicated RMove's architecture and compared its performance against a modified version that incorporated a basic feature fusion layer instead of concatenation. As shown in Table I, applying feature fusion yielded substantial improvements across key metrics, including a **6.2%** increase in F1-score. This result strongly indicates that addressing feature redundancy is critical for improving recommendation accuracy. Therefore, this finding was a primary motivation for designing **GMove** with a sophisticated, multi-branch feature fusion architecture at its core, moving beyond simple concatenation to more effectively integrate diverse feature types.

TABLE I
Comparison between RMove with Feature Fusion

Models	Accuracy	Precision	Recall	F1
Original RMove	83.3%	80.2%	89.1%	84.3%
RMove with feature fusion	90.5%	91.3%	89.8%	90.5%

JMove Dependency Similarity Limitation: JMove's recommendation strategy is predicated on the similarity of dependencies, suggesting a method move to the class with which it shares the most dependencies. However, this heuristic can be misleading. Our analysis of the Maven project (version 3.0.5, from Qualitas.class) revealed a clear example of this limitation. The method 'createBuildContext()', originally in the 'MavenSession' class, is a better fit for the 'LifecycleWeaveBuilderTest' class to improve encapsulation and test maintainability.

JMove fails to recommend this refactoring because the method's dependency similarity with its original class, 'MavenSession', is higher than with the correct target class, 'LifecycleWeaveBuilderTest' ($Similarity = 0.21$ vs. 0.13). Consequently, JMove incorrectly suggests the method should stay in its original class. Source code for this illustrative example is available in our public repository.

Subsequent testing involved assessing missed methods by JMove through RMove. RMove effectively recommended 10 of the 11 methods overlooked by JMove. These outcomes complement the analytical insights of JMove in recognizing refactoring opportunities, resulting in successful suggestions for *Move Method*. The fusion of both methodologies highlights the importance of incorporating semantic features, metric data, and heuristic rules to maximize the effectiveness of static and dynamic analyses. This methodology boosts the precision and effectiveness of detecting software refactoring.

[0]The method createBuildContext() should be moved from MavenSession to LifecycleWeaveBuilderTest. However, JMove fails to recommend this, as the dependency similarity score is higher with the original class (0.21) than the target class (0.13).

III. Design

Fig.1 presents an overview of the GMove pipeline that aims to automatically suggest Move Method refactoring candidates. GMove is structured into three phases: (1) data collection and preparation, where a large-scale balanced data set is constructed; (2) multi-modulal feature representation, where semantic, structural and metric features are systematically extracted and encoded; and (3) Hybrid Deep Learning Model, where a specialized neural network architecture fuses these features to make a final recommendation. Each phase is designed to address specific limitations identified in previous work, such as data scarcity and the feature redundancy inherent in simpler concatenation-based models.

A. Data Collection and Preparation

Deep learning models require substantial labeled data. To address this, we selected 1,000 high-quality Java projects based on four criteria: (1) Active maintenance: ¿10 commits in the last year; (2) Domain diversity: Covering 12 domains (e.g., web, database, GUI); (3) Code maturity: ¿10,000 LOC and ¿50 stars; (4) Documentation quality: Comprehensive README and API docs.

Using MoveMethodGenerator [15], we processed these projects to generate *Move Method* instances. The tool identified movable methods and target classes while filtering non-movable ones (static methods, constructors).

Analysis covered 454 projects (45.4%), discovering 16,828 moveable methods. Training data generation: For each sample, we:

1) Extract method m_i, source class sc_i, and target class tc_i
2) Create negative sample: fuse embeddings of m_i and sc_i (label: False)
3) Create positive sample: fuse embeddings of m_i and tc_i (label: True)

B. Multi-Modal Feature Representation

To create a holistic view of each refactoring candidate, GMove engineers three distinct types of features.

(1) Semantic Feature Representation: We parse each method into an Abstract Syntax Tree (AST) [16] and extract paths between its leaf nodes. We employ a pre-trained Code2Vec [17] model to convert these AST paths into 128-dimensional vector embeddings, capturing the code's functional semantics.

(2) Structural Feature Representation: We build a Method Dependency Graph (MDG) [18] to model the invocation relationships between all methods in a project. We then use Structural Deep Network Embedding (SDNE) [19], a semisupervised deep model, to generate 128-dimensional graph embeddings from the MDG, preserving both local and global network structures.

(3) Metric Feature Representation: We use SourceMeter to extract a comprehensive set of method-level and class-level metrics. To select the most impactful features, we apply Recursive Feature Elimination with Cross-Validation (RFECV) [20]. This process identified 29 relevant method-level and 52 class-level metrics in categories such as cohesion, complexity, and coupling.

C. Hybrid Deep Learning Model for Fusion

To address feature redundancy, we propose a feature fusion method combining semantic, structural, and metric features. We design a deep learning model with a Bi-LSTM [21] branch and two CNN [22] branches to process these features.

Bi-LSTM captures bidirectional semantic dependencies in code snippets, analyzing context from both forward and backward directions. CNNs extract local features through convolution operations, reducing embedding dimensions while retaining essential structural information. This multi-branch architecture enables GMove to leverage diverse features, enhancing accuracy and generalization.

1) Semantic Feature: Semantic information is processed by a Bi-LSTM branch comprising:

a. Input Layer: Receives 256-dimensional code embeddings formed by concatenating 128-dimensional method embeddings $codeEmb(m_i)$ and class embeddings $ClassCE(C_i)$, where $ClassCE(C_i)$ is the average embedding of methods within C_i. Crucially, to strictly prevent data leakage during training and cross-validation, $ClassCE(C_i)$ for any class C_i is calculated exclusively by averaging the embeddings of methods belonging to C_i that are present in the current training set split.

$$MethodCE = \{(m_i, codeEmb\,(m_i))\,|m_i \in MoveMethod_i\} \tag{1}$$

$$ClassCE = \left\{ \left(C_i, \frac{\sum_{m_j \in C_i} codeEmb\,(m_j)}{|C_i|} \right) |C_i \in MoveMethod_i \right\} \tag{2}$$

b. Bi-LSTM Layer: Captures bidirectional semantic dependencies and long-range contextual information.

c. FCNN Layer: Linearly combines Bi-LSTM outputs to extract key features, producing a 128-dimensional semantic feature CE.

2) Structural Feature: Structural information is processed by a CNN_GE branch, which is designed to capture local structural patterns and dependencies within the graph embeddings:

a. Input Layer: Receives 256-dimensional graph embeddings formed by concatenating method graph embeddings $graphEmb(m_i)$ and class graph embeddings $ClassGE(C_i)$, where $ClassGE(C_i)$ is the average graph embedding of methods in C_i. Similarly, $ClassGE(C_i)$ is strictly calculated based on methods belonging to class C_i found exclusively within the training data of the current split, ensuring strict adherence to the data separation principle.

b. CNN Architecture: The CNN_GE branch is implemented as a one-dimensional convolutional neural network designed to process the flattened graph embedding vectors. The architecture consists of the following sequential layers:

- A 1D convolutional layer with 128 output channels, a kernel size of 6, and 'same' padding. This layer takes the 256-dimensional input and transforms it into a 128-channel feature map.
- A Rectified Linear Unit (ReLU) activation function.
- A 1D batch normalization layer for stabilizing and accelerating the training process.

- A second 1D convolutional layer, also with 128 output channels, a kernel size of 6, and 'same' padding, followed by another ReLU activation. This layer further processes the features extracted by the first layer.

The use of two convolutional layers with non-linear activations allows the model to capture complex, hierarchical patterns within the structural embeddings. The 'same' padding preserves the length of the input sequence throughout the convolutional layers, ensuring no information is lost at the boundaries before flattening.

c. Output Processing: The output from the second convolutional layer is flattened into a single vector. This vector serves as the final 128-dimensional structural feature representation GE. This dimension was chosen to balance information retention and model complexity.

$$MethodGE = \{(m_i, graphEmb\,(m_i))\,|m_i \in MoveMethod_i\} \quad (3)$$

$$ClassGE = \left\{ \left(C_i, \frac{\sum_{m_j \in C_i} graphEmb\,(m_j)}{|C_i|} \right) |C_i \in MoveMethod_i \right\} \quad (4)$$

3) Metric Feature: The selected metrics (after RFECV selection and normalization) are processed by a dedicated CNN_ME branch. The input dimension of 81 corresponds to the total number of selected method-level and class-level metrics (29 + 52). To ensure consistent and effective feature extraction across different data modalities, we adopted an architecture for CNN_ME that is identical in structure to the CNN_GE branch described in Section 3.4.2. This design choice allows the model to process the numerical metric vector in a manner analogous to how it processes graph embeddings, leveraging the same proven capabilities of the 1D-CNN structure for capturing local patterns and hierarchical features.

Specifically, the CNN_ME branch consists of two 1D convolutional layers (each with 128 output channels, a kernel size of 6, and 'same' padding), each followed by a ReLU activation function and a batch normalization layer. The key distinction lies in the input: whereas CNN_GE processes 256-dimensional graph embeddings, CNN_ME takes the 81-dimensional normalized metric vector as its input. The design rationale for applying this CNN architecture to numerical metrics is to automatically model the local correlations and complex interactions between different code metrics, which are often non-linear and can be indicative of refactoring opportunities.

The output from the convolutional layers is flattened into a single vector, forming the final 128-dimensional metric feature representation ME.

4) Deep Learning Model: For the final classification task, we employ a Fully Connected Neural Network (FCNN). This choice is motivated by the FCNN's established effectiveness in mapping diverse feature representations into a more distinguishable space, making it ideal for our fusion-based approach. The FCNN architecture is designed to capture the complex, non-linear interactions between the semantic, structural, and metric features. It comprises three fully connected layers (with 384, 256, and 128 neurons, respectively), each utilizing a ReLU activation function and a dropout layer (rate=0.5) for regularization. The output layer uses a sigmoid activation for binary prediction. We train the model using the Adam optimizer with a binary cross-entropy loss function and implement early stopping (patience=10) to mitigate overfitting.

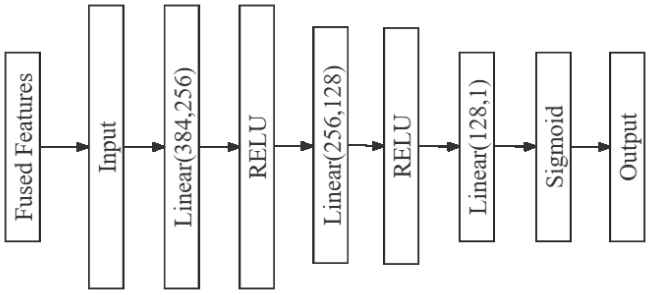

Fig. 2. Deep learning classifier model of *GMove*.

IV. Evaluation

This section first introduces the experimental setup, followed by the research questions and evaluation criteria. Finally, the experimental results are illustrated. All models and data are publicly available at *https://zenodo.org/records/11401052*.

A. Experimental Framework

All experiments are conducted on a workstation with a 2.9 GHz Intel Core i7-10700 CPU and 128 GB main memory, running Ubuntu 22.04 operating system with PyTorch 2.2 installed. The hyperparameters for all models follow the default settings and are shown in Table II.

B. Research Questions

We evaluate the effectiveness of *GMove* by answering the following research questions (RQ).

- RQ1 How effective is the generated dataset compared to the real dataset?
- RQ2 How applicable is *GMove* in recommending *Move Method* refactorings?
- RQ3 How effective is feature fusion in recommending *Move Method* refactorings?
- RQ4 How effective is *GMove* than existing machine learning models and deep learning models in recommending *Move Method* refactorings?
- RQ5 How effectively does *GMove*, trained on the synthetic dataset, generalize in detecting *Move Method* opportunities compared to existing works on a real-world benchmark?
- RQ6 How useful is *GMove* in recommending move method refactoring?

TABLE II
HYPERPARAMETER SETTINGS OF MACHINE LEARNING MODEL AND DEEP LEARNING MODEL

Models	Hyperparameter
Code2Vec	num_epochs=20,train_batch_size=1024,test_batch_size=1024,code_vector_size=128,path_embeddings_size=128 token_embeddings_size=128,csv_buffer_size=100*1024*1024,default_embeddings_size=128
SDNE	alpha=1e-6,beta=5,nu1=1e-5,nu2=1e-4,batch_size=200,epoch=100
XGBoost	learning_rate=0.3,n_estimators=100,max_depth=6
SVC	degree=3,tol=0.001,cache_size=200
Decision Tree	max_depth=None,min_samples_split=2 min_impurity_decrease=0,ccp_alpha=0.0
Random Forest	n_estimators=100
ExtraTrees	n_estimators=100
Naive Bayes	var_smoothing=1e-09
Logistic Regression	tol=0.0001, max_iter=100

C. Evaluation Metrics

We evaluate the effectiveness of *GMove* by calculating several metrics including accuracy, precision, recall, and F1. These metrics can be computed by Equations 5-7.

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

True Positive (TP) denotes the actual number of refactorings required and correctly predicted. True Negative (TN) represents the actual number of refactorings not required and not suggested. False Positive (FP) indicates the actual number of unnecessary refactorings incorrectly suggested, and False Negative (FN) indicates the actual number of refactorings required but not suggested.

D. Results

This section presents the experimental results of the research questions.

1) Results for RQ1: We evaluated MoveMethodGenerator's effectiveness using eight projects from the widely-used Qualitas.class corpus , previously studied in literature [6], [9], [10]. Table III details the projects, including names, versions, LOC, NOC, NOM, and *Move Method* counts. In Table IV, the *Move Method* count represents the set of confirmed, real-world refactoring instances for each project, which were originally manually validated and documented in prior studies. We use the MoveMethodGenerator to create refactoring examples in these eight projects and conduct comparative experiments with real datasets. This is done to verify whether the generated datasets can effectively replace real datasets and to explore their performance in various machine learning and deep learning models. Generated datasets from these projects were compared against real datasets using five machine learning models (DT [23], SVC [24], LR [25], RF [26], ET [27]) and two deep learning models (CNN, GRU).

To ensure a fair comparison, both the Real Dataset (R-Set) and the Generated Dataset (G-Set) were constructed using a 1:1 positive-to-negative sample ratio. The G-Set was constructed by processing the same projects using the MoveMethodGenerator to extract an equal number of positive (method m, target class tc) and negative (method m, non-target class sc) instances, following the procedure in Section 3.1. The R-Set uses the confirmed real-world *Move Method* instances as positives. For R-Set negative samples, we used the MoveMethodGenerator to identify an equal number of non-refactoring candidates from the same projects that do not overlap with the confirmed true positives.

TABLE III
STATISTICAL SUMMARY OF THE EIGHT OPEN-SOURCE JAVA PROJECTS SELECTED FROM THE QUALITAS.CLASS CORPUS FOR EVALUATION. THIS TABLE DETAILS THE PROJECT CONFIGURATIONS USED TO VALIDATE THE MODEL

project	Version	LOC	NOC	NOM	# of *Move Method*
Ant	1.8.2	103402	760	8586	25
DrJava	r5387	88631	361	4675	18
FreeCol	0.10.3	93605	535	6616	17
FreeMind	0.9.0	53782	368	4074	12
JMeter	2.5.1	81222	682	7392	25
JTOpen	7.8	340752	1450	22143	39
Maven	3.0.5	71065	154	1568	24
Weka	3.6.9	257897	908	16034	31

As shown in Table IV, the generated dataset exhibits comparable or superior performance to the real dataset across various models. Notably, in the RF model, the generated dataset achieved a precision of 79.0%, slightly surpassing the real dataset's 77.3%. Furthermore, the LR model showed superior accuracy and F1 (0.738) on the generated data compared to the real data. This confirms the generated data successfully simulates key features and patterns necessary for effective classification.

Summary of RQ1: MoveMethodGenerator generated datasets effectively simulate real data performance across ML/DL models and surpass real data in key metrics.

2) Results for RQ2: To evaluate the applicability of *GMove* for the *Move Method* refactoring, we use the synthetic dataset from Section 3.1 for training and validation of our model, where the ratio of training set, test set, and validation set are 3:1:1, respectively. As shown in the first row of Table V, *GMove* achieves an accuracy of 97% on the validation set,

TABLE IV
PERFORMANCE COMPARISON BETWEEN REAL DATASETS AND GENERATED DATASETS

	RF		DT		ET		LR		*SVC*		CNN		*GRU*	
	Real	Gen	Real	Gen	Real	Gen	Real	Gen	Real	Gen	Real	Gen	Real	Gen
Accurary	**75.4%***	72.6%	**66.7%***	65.5%	**69.3%***	64.9%	72.8%	**73.8%***	50.9%	**51.2%***	**71.9%***	69.1%	43.9%	**49.4%***
Precision	77.3%	**79.0%***	66.1%	**73.3%***	**75.0%***	71.8%	**73.2%***	72.1%	51.1%	**53.1%***	**70.8%***	70.5%	44.7%	**49.4%***
Recall	**71.9%***	68.8%	**68.4%***	66.0%	57.9%	**60.2%***	71.9%	**75.6%***	40.4%	**75.6%***	**78.0%***	75.5%	35.6%	**52.4%***
F1	**74.5%***	73.5%	67.2%	**69.5%***	65.3%	**65.5%***	72.5%	**73.8%***	45.1%	**62.4%***	**74.2%***	72.9%	39.6%	**50.9%***

a precision of 98.4%, a recall of 97.1%, and an F1 score of 97.7%. These results demonstrate that *GMove* exhibits strong performance on the synthetic dataset.

Summary of RQ2: GMove demonstrates high applicability and effectiveness on a large-scale synthetic dataset (97.7% F1). This confirms the viability of our proposed deep learning architecture as a solution for the Move Method recommendation task.

3) Results for RQ3: To evaluate feature fusion's effectiveness in recommending *Move Method* refactorings, we systematically compared combinations of semantic (CE), structural (GE), and metric (ME) features. We fused features using splicing and point-wise weighted average and employed identical-depth fully connected networks as classifiers. Table V illustrates the performance of various feature combinations on synthetic datasets. The CE+GE feature combination is used in RMove [10]. We use a grey background to highlight the model with the best performance and use * to mark the maximum scores for Accuracy, Precision, Recall, and F1.

TABLE V
THE PERFORMANCE OF VARIOUS FEATURE COMBINATIONS ON THE SYNTHETIC DATASET. CE: SEMANTIC FEATURES, GE: STRUCTURAL FEATURES, ME: METRIC FEATURES, MEAN: POINT-WISE WEIGHTED AVERAGE

Models	Accuracy	Precision	Recall	F1
CE+GE+ME	**97%***	**98.4%***	**97.1%***	**97.7%***
CE+GE+ME_Mean	88.9%	88%	90.4%	89.2%
CE+GE(RMove)	83.3%	80.2%	89.1%	84.3%
CE+GE_Mean	90.5%	91.3%	89.8%	90.5%
CE+ME	71.2%	70%	78%	73.7%
CE+ME_Mean	70.8%	74.3%	64.6%	69.1%
GE+ME	90.8%	91.9%	93.7%	92.4%
GE+ME_Mean	90.8%	90.6%	91.4%	91%

In Table V, we observed that the CE+GE+ME combination achieved the best results (97%, 98.4%, 97.1%, 97.7%), highlighted in grey. GE+ME also outperformed other combinations. CE+GE+ME_Mean underperformed, likely due to feature exclusion during averaging and limited model parameters.

CE+GE+ME's superiority stems from integrating comprehensive semantic, structural, and metric information. GE+ME's effectiveness highlights the importance of structural features combined with method-level specifics for *Move Method* recommendations, surpassing semantic-only approaches.

Summary of RQ3: The fusion of semantic, structural, and metric features (CE+GE+ME) is the most effective combination, outperforming other variants. This result directly validates our core contribution regarding the superiority of a multi-modal feature fusion strategy.

4) Results for RQ4: To evaluate RQ4, we ranked the RQ2 model combinations on real-world datasets, selecting the best-performing combination CE+GE+ME as *GMove*. We compared *GMove* against seven common machine learning models (DT [23], NB [28], SVC [24], LR [25], RF [26], XGB, ET [27]) and two deep learning models (CNN, GRU). Performance is shown in Table VI. The best-performing model is highlighted in grey, and maximum scores among the 7 ML models are marked with an asterisk.

Feature consistency: All compared models used identical features as *GMove*, ensuring a fair architectural comparison.

TABLE VI
PERFORMANCE OF GMOVE'S HYBRID ARCHITECTURE VS. STANDARD MODELS. GMOVE SUBSTANTIALLY OUTPERFORMS BOTH TRADITIONAL ML CLASSIFIERS AND GENERIC DL MODELS, HIGHLIGHTING THE BENEFIT OF ITS SPECIALIZED FUSION DESIGN

Models	Accuracy	Precision	Recall	F1
XGB	74.2%	73.2%	72%	**72.6%***
SVC	47.2%	47.4%	**99.4%***	64.2%
Decision Tree	62.5%	59.3%	67.3%	63%
Random Forest	68.9%	67.5%	66.7%	67.1%
ExtraTrees	65.6%	64.4%5	61.4%	62.9%
Naive Bayes	55%	63.6%	12.3%	20.6%
Logistic Regression	64.4%	62%	64.9%	63.4%
CNN	75.6%	73.2%	67.1%	70.0%
GRU	**77.6%***	**78.7%***	65.2%	71.3%
GMove	**97%**	**98.4%**	**97.1%**	**97.7%**

Table VI shows *GMove* excels across all metrics, achieving 97% accuracy, 98.4% precision, 97.1% recall, and 97.7% F1. Compared to the ML models, *GMove* significantly improves accuracy by 22.8%, precision by 25.2%, and F1 by 25.1%. While SVC achieved a slightly higher recall (99.4%) than *GMove*'s 97.1%, its advantage likely stems from its maximum-margin design, enhancing recall at the cost of significantly lower precision (47.4%).

Despite SVC's higher recall, *GMove* substantially outperforms it in accuracy (+49.8%), precision (+51%), and F1 (+33.5%). Overall, *GMove* demonstrates superior performance to both ML and deep learning models.

Summary of RQ4: GMove's specialized deep learning model significantly outperforms seven traditional machine learning classifiers and two standard deep learning models using the same fused features. This underscores the novelty and effectiveness of our custom-designed hybrid neural network architecture.

TABLE VII
END-TO-END COMPARISON OF GMOVE WITH STATE-OF-THE-ART REFACTORING TOOLS ON THE QUALITAS.CLASS BENCHMARK. GMOVE ACHIEVES THE BEST BALANCE OF PRECISION AND RECALL, LEADING TO THE HIGHEST OVERALL F1-SCORE IN NEARLY EVERY PROJECT

Subject	PathMove			JDeodorant			JMove			RMove		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Weka	22.4%	64.5%	33.2%	5.9%	54.8%	10.7%	10.8%	74.1%	18.9%	50.8%	95.9%	66.4%
Ant	19.7%	56.0%	29.2%	17.1%	48.0%	25.2%	17.3%	84.0%	28.7%	49.1%	90.5%	63.7%
FreeCol	4.8%	64.7%	8.9%	3.0%	29.4%	5.4%	7.4%	76.4%	13.5%	48.3%	87.9%	62.3%
Jmeter	26.4%	36.0%	30.5%	23.6%	52.0%	32.5%	27.5%	76.0%	40.4%	46.6%	92.3%	62.0%
FreeMind	80.0%	33.3%	47.0%	16.6%	58.3%	25.8%	14.8%	66.6%	24.2%	51.7%	**96.4%**	67.3%
JOpen	41.6%	51.2%	45.9%	20.7%	44.7%	28.3%	20.8%	89.4%	33.7%	48.4%	87.2%	62.3%
DrJava	42.8%	50.0%	46.1%	12.8%	55.5%	20.8%	12.8%	77.7%	21.9%	52.6%	**86.0%**	68.0%
Maven	54.5%	54.1%	54.3%	13.9%	25.0%	17.9%	10.4%	37.5%	16.3%	49.6%	85.3%	62.7%
Avg	36.5%	51.2%	36.9%	14.2%	46.0%	20.8%	15.2%	72.7%	24.7%	49.6%	91.4%	64.3%
Subject	FeTruth			FePM			HMove			Gmove		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Weka	42.1%	75.8%	54.3%	69.5%	46.8%	56.0%	**92.5%**	71.8%	80.8%	75.0%	**96.4%**	**84.3%**
Ant	44.6%	69.1%	54.2%	72.1%	48.2%	57.8%	**97.5%**	66.6%	79.1%	76.7%	**95.8%**	**85.2%**
FreeCol	38.5%	72.6%	50.2%	62.0%	40.5%	49.0%	**91.2%**	67.5%	77.6%	77.8%	**93.4%**	**84.8%**
Jmeter	36.8%	78.9%	51.1%	60.8%	32.2%	43.0%	**87.4%**	63.7%	72.3%	82.1%	**95.8%**	**88.4%**
FreeMind	40.2%	74.3%	52.1%	64.2%	45.9%	53.5%	**90.8%**	68.9%	78.3%	78.6%	91.7%	**86.6%**
JTopen	35.7%	77.4%	48.9%	70.1%	44.2%	54.2%	**96.3%**	55.0%	70.0%	84.8%	**86.6%**	**90.3%**
DrJava	30.3%	83.3%	44.4%	68.6%	44.2%	53.8%	**96.4%**	66.7%	78.8%	73.7%	93.4%	**82.3%**
Maven	37.2%	76.5%	49.8%	64.8%	36.1%	46.5%	**91.7%**	63.2%	74.8%	87.5%	**93.3%**	**89.5%**
Avg	38.2%	76.0%	50.6%	66.5%	42.4%	51.7%	**93.0%**	65.4%	76.5%	79.5%	**94.6%**	**86.2%**

5) Results for RQ5: To investigate the effectiveness of *GMove* in recommending *Move Method* refactoring and specifically to address its real-world generalizability (transfer effectiveness), we compare it against the state-of-the-art refactoring approaches, such as PathMove [9], JDeodorant [7], JMove [6], RMove [10], FeTruth [13], FePM [12] and HMove [11]. To ensure the reliability of our experimental results, we selected the open source dataset Qualitas.class [29] corpus. The Qualitas.class corpus has been widely used in previous research [6], [9], [10]. Evaluation results are presented in Table VII. We use a grey background to mark the maximum scores of Precision, Recall, and F1 for each project.

Here is a focused analysis comparing GMove against key baselines:

- JMove (Avg P: 15.2%, Avg R: 72.7%, Avg F1: 24.7%) and PathMove (Avg P: 36.5%, Avg R: 51.2%, Avg F1: 36.9%): These heuristic-based methods rely on predefined rules (dependency similarity or path-based representations), which fundamentally fail to capture the complex, nuanced code semantics and structures needed for accurate recommendations. Their low average Precision and F1s confirm that they are less robust than deep learning methods, often suggesting invalid refactorings.
- RMove (Avg P: 49.6%, Avg R: 91.4%, Avg F1: 64.3%): RMove's high Recall (91.4%) is strong, but its lower Precision (49.6%) and F1 are limited by feature redundancy due to simple concatenation of features. GMove addresses this via feature fusion, resulting in superior performance across the board.
- FeTruth (Avg P: 38.2%, Avg R: 76.0%, Avg F1: 50.6%) and FePM (Avg P: 66.5%, Avg R: 42.4%, Avg F1: 51.7%): These models utilize powerful modern semantic representations from pre-trained language models (like CodeT5 for FePM), but they are significantly outperformed by GMove by over 34% in F1. This confirms that a comprehensive feature fusion of semantic, structural, and metric information is superior to relying primarily on textual features alone for the Move Method task.
- HMove (Avg P: 93.0%, Avg R: 65.4%, Avg F1: 76.5%): HMove, which uses a complex Hypergraph Neural Network, achieves the highest average Precision (93.0%) among all tools, indicating that its graph modeling excels at ensuring recommendations are valid. However, GMove maintains a 9.7 percentage point higher F1 (86.2% vs. 76.5%) because HMove's overly conservative graph approach leads to lower Recall (65.4%), whereas GMove's balanced feature fusion provides a better trade-off between minimizing false positives and false negatives.

To rigorously validate the significance of these results, we first performed an ANOVA test, which indicated a significant difference among the tools' performances (F = 77.34, p ¡ 0.001). We then conducted a Tukey HSD post-hoc test for pairwise comparisons. The results confirm that **GMove's F1 score is statistically significantly higher than all seven state-of-the-art baseline tools (p ¡ 0.001 for all comparisons)**, lending strong statistical support to our claims of superiority.

Summary of RQ5: GMove achieves a state-of-the-art F1-score of 86.2%, outperforming the strongest baseline by 9.7%. This finding directly substantiates our second major contribution: delivering a demonstrably more effective solution than existing tools.

6) Results for RQ6: To evaluate the practical application of GMove, we conducted a study with six software engineers, none of whom contributed to this paper. Participants reviewed

detection results from five refactoring tools on Ant (from RQ5), provided with anonymized, randomized samples of five instances per tool. For each tool, participants rated: (1) Likelihood of use (1-5 scale) and (2) Perceived accuracy (1-3 scale).

TABLE VIII
PARTICIPANTS' RATINGS FOR EACH REFACTORING TOOL

Participants/Total Score	PathMove		JDeodorant		JMove		RMove		*GMove*	
	RQ1	RQ2	RQ1	RQ2	RQ1	RQ2	RQ1	RQ2	RQ1	RQ2
P1	2	2	1	1	3	2	**5**	**3**	4	**3**
P2	1	1	2	2	4	**3**	3	2	**5**	**3**
P3	3	2	4	2	**5**	3	**5**	**3**	**5**	**3**
P4	3	1	1	1	4	2	4	2	2	2
P5	3	2	2	1	1	1	**5**	**3**	**5**	**3**
P6	**5**	**3**	3	2	4	**3**	2	**3**	**5**	**3**
Total Score	17	11	13	9	21	14	24	16	**26**	**17**

Table VIII presents the survey results from six participants. (1) GMove achieved the highest total scores in both usability (26) and accuracy (17). (2) Five participants rated GMove $\geq$ 4 for usability, and all rated its accuracy $\geq$ 2. This indicates GMove received higher recognition than state-of-the-art alternatives.

Summary of RQ6: In a study with software engineers, GMove was rated highest in both perceived accuracy and likelihood of use, indicating that our technical contributions translate into a tool that practitioners find genuinely useful and trustworthy. However, we recognize that this preliminary study with a small sample size ($N = 6$) is qualitative and not statistically conclusive. We address the plan for a large-scale validation in the Limitations and Future Work section.

E. Limitations and Future Work

This study has four main limitations that inform our future work:

1) Language Scope: GMove is currently limited to Java. Future work will extend the framework to other languages like Python and C# to validate its cross-language applicability.
2) Data Source: The model relies on a synthetically generated dataset, which may introduce biases. We plan to mitigate this by creating a large, human-annotated dataset and using it to fine-tune the model. To further address real-world utility, we will explore the integration of *GMove* into IDE plugins and CI/CD pipelines for validation in practical developer workflows.
3) Model Interpretability: As a deep learning model, GMove operates as a "black box," which can hinder developer trust. We will integrate explainability techniques (e.g., SHAP, LIME) to provide transparent, human-readable justifications for its recommendations. Furthermore, we plan to conduct a dedicated user study to assess the practical effectiveness of these explanations in enhancing developer trust and adoption.
4) Practitioner Study Scale: Our initial user study was qualitative with a small sample size (N=6). A future large-scale study (N $\geq$ 60) will be conducted to statistically validate the tool's practical utility. We commit to making this large-scale, statistically robust user study a high-priority future work item based on reviewer feedback.

V. CONCLUSION

This paper presents a novel approach called *GMove* for recommending *Move Method* refactoring opportunities. We constructed a large-scale synthetic dataset and leveraged feature fusion to combine semantic, structural, and metric features. The features are processed and fused by a hybrid deep learning model with *Bi-LSTM* and *CNN* branches, with final classification performed by *FCNN*. Experimental results show that the F1 of *GMove* is 97.7% on the synthetic dataset. Furthermore, comparison on a publicly available real-world dataset demonstrates the effectiveness of *GMove*, achieving an F1 improvement of up to 65.4% compared to existing refactoring tools. Future work includes exploring the potential influence of hyperparameters on embedding techniques and automatically generating *Move Method* instances for various programming languages.

ACKNOWLEDGMENTS

This work is partially supported by the Natural Science Foundation of Hebei Province under Grant F2023208001, the Oversea High-level Talent Foundation of Hebei Province under Grant C20230358, Central Guidance on Local Science and Technology Development Fund under Grant 254Z1601G.

REFERENCES

[1] M. Fowler, "Refactoring," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 701. [Online]. Available: https://doi.org/10.1145/581339.581453

[2] Y. Zhang, C. Ge, H. Liu, and K. Zheng, "Code smell detection based on supervised learning models: A survey," *Neurocomputing*, vol. 565, p. 127014, 2024. [Online]. Available: https://doi.org/10.1016/j.neucom.2023.127014

[3] L. Chen and S. Hayashi, "Impact of change granularity in refactoring detection," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, ser. ICPC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 565–569. [Online]. Available: https://doi.org/10.1145/3524610.3528386

[4] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 347–367, 2009. [Online]. Available: https://doi.org/10.1109/TSE.2009.1

[5] Y. Zhang, K. Guan, and L. Fang, "Mirror: multi-objective refactoring recommendation via correlation analysis," *Automated Software Engineering*, vol. 31, no. 1, p. 2, 2024. [Online]. Available: https://doi.org/10.1007/s10515-023-00400-1

[6] R. Terra, M. T. Valente, S. Miranda, and V. Sales, "Jmove: A novel heuristic and tool to detect move method refactoring opportunities," *Journal of Systems and Software*, vol. 138, pp. 19–36, 2018. [Online]. Available: https://doi.org/10.1016/j.jss.2017.11.073

[7] M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou, "Jdeodorant: Identification and removal of feature envy bad smells," in *2007 IEEE International Conference on Software Maintenance*, 2007, pp. 519–520. [Online]. Available: https://doi.org/10.1109/ICSM.2007.4362679

[8] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "Methodbook: Recommending move method refactorings via relational topic models," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 671–694, 2014. [Online]. Available: https://doi.org/10.1109/TSE.2013.60

[9] Z. Kurbatova, I. Veselov, Y. Golubev, and T. Bryksin, "Recommendation of move method refactoring using path-based representation of code," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 315–322. [Online]. Available: https://doi.org/10.1145/3387940.3392191

[10] D. Cui, S. Wang, Y. Luo, X. Li, J. Dai, L. Wang, and Q. Li, "Rmove: Recommending move method refactoring opportunities using structural and semantic representations of code," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2022, pp. 281–292. [Online]. Available: https://doi.org/10.1109/ICSME55016.2022.00033

[11] D. Cui, J. Wang, Q. Wang, P. Ji, M. Qiao, Y. Zhao, J. Hu, L. Wang, and Q. Li, "Three heads are better than one: Suggesting move method refactoring opportunities with inter-class code entity dependency enhanced hybrid hypergraph neural network," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 745–757. [Online]. Available: https://doi.org/10.1145/3691620.3695068

[12] W. Ma, Y. Yu, X. Ruan, and B. Cai, "Pre-trained model based feature envy detection," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 2023, pp. 430–440. [Online]. Available: https://doi.org/10.1109/MSR59073.2023.00065

[13] B. Liu, H. Liu, G. Li, N. Niu, Z. Xu, Y. Wang, Y. Xia, Y. Zhang, and Y. Jiang, "Deep learning based feature envy detection boosted by real-world examples," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 908–920. [Online]. Available: https://doi.org/10.1145/3611643.3616353

[14] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," 2021. [Online]. Available: https://arxiv.org/abs/2109.00859

[15] E. Novozhilov, I. Veselov, M. Pravilov, and T. Bryksin, "Evaluation of move method refactorings recommendation algorithms: Are we doing it right?" in *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*, 2019, pp. 23–26. [Online]. Available: https://doi.org/10.1109/IWoR.2019.00012

[16] V. Kovalenko, E. Bogomolov, T. Bryksin, and A. Bacchelli, "Pathminer: A library for mining of path-based representations of code," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 13–17. [Online]. Available: https://doi.org/10.1109/MSR.2019.00013

[17] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "Code2vec: Learning distributed representations of code," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, jan 2019. [Online]. Available: https://doi.org/10.1145/3290353

[18] H. Cai and R. Santelices, "Abstracting program dependencies using the method dependence graph," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 49–58. [Online]. Available: https://doi.org/10.1109/QRS.2015.18

[19] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1225–1234. [Online]. Available: https://doi.org/10.1145/2939672.2939753

[20] A. Z. Mustaqim, S. Adi, Y. Pristyanto, and Y. Astuti, "The effect of recursive feature elimination with cross-validation (rfecv) feature selection algorithm toward classifier performance on credit card fraud detection," pp. 270–275, 2021. [Online]. Available: https://doi.org/10.1109/ICAICST53116.2021.9497842

[21] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997. [Online]. Available: https://doi.org/10.1109/78.650093

[22] S. Skansi, "Convolutional neural networks," pp. 121–133, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-73004-2_6

[23] S. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660–674, 1991. [Online]. Available: https://doi.org/10.1109/21.97458

[24] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006. [Online]. Available: https://doi.org/10.1038/nbt1206-1565

[25] H. C. Jessen, "Applied logistic regression analysis," *Journal of the Royal Statistical Society Series D: The Statistician*, vol. 45, no. 4, pp. 534–535, 12 2018. [Online]. Available: https://doi.org/10.2307/2988559

[26] M. Pal, "Random forest classifier for remote sensing classification." vol. 26, p. 217–222, 2005. [Online]. Available: https://doi.org/10.1080/01431160412331269698

[27] J. Simm and I. Abril, "Extratrees: extremely randomized trees (extratrees) method for classification and regression," *R package version*, vol. 1, no. 5, 2014. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1

[28] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng, "Some effective techniques for naive bayes text classification," vol. 18, no. 11, 2006, pp. 1457–1466. [Online]. Available: https://doi.org/10.1109/TKDE.2006.180

[29] R. Terra, L. F. Miranda, M. T. Valente, and R. S. Bigonha, "Qualitas.class corpus: a compiled version of the qualitas corpus," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 5, p. 1–4, Aug. 2013. [Online]. Available: https://doi.org/10.1145/2507288.2507314

Yang Zhang received the Ph.D. degree in computer software and theory from Beijing Institute of Technology, Beijing, China. He is a Professor and a Muxing Scholar with Hebei University of Science and Technology, Shijiazhuang, China. His research interests include software refactoring, intelligent software, and parallel programming.

Zhenggang Gu is currently pursuing his master's degree in the School of Information Science and Engineering, Hebei University of Science and Technology. His research interests include software testing and software refactoring.

Nan Zhang is an intermediate engineer and deputy manager of the Policy Research and Planning Office, holding a Master's degree. Her research interests include enterprise informatization, intelligentization, and digitalization.

Kun Zheng is currently an associate professor at Hebei University of Science and Technology. Her research interests include intelligent software and code refactoring.