Machine Learning for Human Intelligence
2026, Vol. 4, Issue 1

Research Article

A Self-Supervised Semantic Representation Learning Framework with Adaptive Feature Fusion for Cross-Project Software Defect Prediction

Muhammad Zain Waqar[1,⋆], Armaghan Mubeen Butt[1], and Muhammad Faheem Khan[2]

[1]Department of Computer Science, The University of Alabama at Birmingham, United States.; Email: zayaanwaqar@gmail.com ; armaghanbutt50@gmail.com
[2]Department of Computer Science, TIMES Institute, Multan, 60000, Pakistan.; Email: mfk7733@gmail.com
⋆Corresponding author: Muhammad Zain Waqar (zayaanwaqar@gmail.com)

Article History

Academic Editor:
Dr. Muhammad Sajid
Submitted: August 26, 2025
Revised: February 09, 2026
Accepted: March 01, 2026

Keywords:
Software Vulnerability, Software Defects, Self-Supervised Learning, Defect Prediction, Semantic Representation Learning.

Abstract

Cross-Project Software Defect Prediction (CPDP) has become a promising research direction for enhancing software reliability when labeled defect samples are scarce or absent in real-world situations. Nevertheless, current CPDP models frequently encounter domain shift, inconsistent feature distributions, and suboptimal generalizability across heterogeneous software projects. This paper presents a new Self-Supervised Semantic Representation Learning approach that combines Adaptive Feature Fusion and Cross-Project Domain Alignment to address the aforementioned issues in cross-project defect prediction. The proposed technique is composed of two parts: (i) self-supervised semantic learning using a Transformer-based contrastive learning model to generate reusable semantic embeddings of unlabeled software repositories; and (ii) an adaptive feature fusion layer to combine these semantic embeddings with handcrafted software metrics. While the conventional CPDP framework primarily relies on handcrafted measures to learn semantic representations, our model leverages an invariant semantic relation among software entities through contrastive optimization. Moreover, cross-project feature alignment and transfer fine-tuning techniques have been applied to alleviate the problem of heterogeneous distribution in the latent space. We experimentally compared the proposed approach with current baseline models using the well-known AEEEM and PROMISE datasets. It was found that the proposed framework achieved state-of-the-art prediction accuracy, yielding Recall, F1-score, MCC, and AUC metrics of 0.93, 0.91, 0.81, and 0.96, respectively. Our study showed through extensive experiments and statistical analysis that incorporating self-supervised semantic learning, adaptive feature fusion, and cross-project domain alignment could significantly improve the prediction robustness and minimize false negative defects.

1 Introduction

The problem of software defects has remained one of the most pressing problems in software engineering since software bugs can adversely affect the reliability, maintainability, and security of the software.

In view of the increasing complexity of software applications, the need for precise and massive bug data sets has risen sharply to facilitate deep learning approaches for software maintenance and defect detection. Many of the existing bug data sets can be either precise with limited size or massive yet less accurate owing to ineffective verification methods. In order to address these issues, some researchers have proposed state-of-the-art bug collection and injection frameworks that can create precise and project-specific bug data sets in multiple programming languages [1].

Due to the ever-growing complexity of software applications, identifying defective software modules during early development stages has become increasingly difficult. Software modules carrying defects that are not identified during the testing processes can end up being deployed, which could result in system failure, financial damage, and software security risks. In this regard, Software Defect Prediction (SDP) has become a highly relevant topic of research concerning quality improvement in software and reducing debugging activities. Although many Deep learning architectures has been developed like TabNet, Neural Oblivious Decision Ensembles (NODE) and FT transformers for enhancing the accuracy in prediction of defects in software [2]. Traditional techniques used in software defect prediction apply machine learning algorithms based on defect repositories, together with software metrics such as cohesion, coupling, complexity, inheritance, and size-based characteristics. Although these methods have shown good results within project predictions, they usually face challenges in Cross-Project Software Defect Prediction (CPDP) [3]. For instance, in industry settings, new projects usually do not have enough labeled defects to allow the use of within-project learning-based prediction. This problem is tackled through cross-project defect prediction.

Although considerable advancements have been achieved in CPDP studies, multiple issues still restrict the robustness and generalization ability of existing techniques. Among these challenges, domain shift between source and target projects remains one of the most critical problems. Software projects commonly vary in coding standards, architectural designs, software evolution behaviors, metric distributions, and defect patterns. Consequently, prediction models trained on one project frequently perform poorly on unseen projects because the extracted feature representations remain highly project-dependent instead of transferable across domains. In addition, conventional machine learning techniques rely extensively on handcrafted software metrics, which may fail to capture complex semantic relationships embedded within software modules [4]. However, recent advances in deep learning, such as transformers and pre-trained language models, have been able to capture and represent semantic information in software. In addition, recent advances in self-supervised representation learning have been able to achieve meaningful and stable results without the need for large amount of labeled data [5]. For instance, recent advances in contrastive learning, which is one of the types of self-supervised representation learning techniques, have been able to learn highly invariant and discriminative representations by maximizing similar sample pairs while minimizing dissimilar sample pairs [6, 7].

Recently, more attention has been devoted in software engineering to the use of deep learning and semantic modeling methods in software defects prediction, vulnerability detection, and commit intent classification tasks. One such work used a transformer model based on the CodeBERT and GraphCodeBERT models for software commit level intent classification. In this work, semantic feature representation has been derived from code-level differences and then aggregated to a higher-level unified representation of commits. As a result, the performance of the intent category classification task has been remarkably improved [8]. For another task, i.e., cross-project software vulnerability detection, the DAP2ER model used a novel progressive domain adaptation approach combining entropy regularization, adversarial learning, and prototype pseudo label optimization in order to solve domain shift and class imbalance problems, which yielded a considerable improvement in F1-score and AUC metric values [9]. Moreover, an LSTM semantic modeling approach was presented that combined AST feature representation with SMOTE and cost-sensitive learning algorithms to better capture context dependence for software defects prediction [10]. Early methods used for defect prediction were mostly based on conventional software metrics such as code size and complexity. However, such metrics are not sufficient enough to capture the semantic relationships present in the source code. Although the newer methods have introduced the idea of extracting semantic features from the source code, using only one set of such features adversely impacts the performance of defect prediction. Thus, combining

both sets of features has become necessary to achieve accurate defect prediction [11].

In light of the above shortcomings, this paper proposes a novel Self-Supervised Semantic Representation Learning Framework with Adaptive Feature Fusion and Cross-Project Domain Alignment to predict software defects across multiple heterogeneous software projects. Specifically, the new approach is based on a transformer-based self-supervised contrastive learning paradigm that will enable direct learning of invariant software representations. In other words, unlike the conventional supervised learning approach, the proposed model will be able to learn generic representations of software semantics irrespective of projects. The second contribution of this study includes the development of an adaptive feature fusion layer. As part of this new paradigm, the proposed framework dynamically combines semantic embeddings with handcrafted software metrics to derive a unified feature representation. By doing so, the framework is expected to capture both semantic contextual information as well as software structure. In addition, the new framework incorporates cross-project feature domain alignment to mitigate any possible distribution mismatch and enhance generalizability. The major contributions of this study are summarized as follows:

1. A novel self-supervised semantic representation learning framework is proposed for Cross-Project Software Defect Prediction using Transformer-based contrastive learning.

2. An Adaptive Feature Fusion Layer is introduced to dynamically integrate semantic embeddings and handcrafted software metrics for improved defect-aware representation learning.

3. A cross-project domain alignment strategy is developed to reduce feature distribution discrepancy between heterogeneous software projects.

4. Extensive experimental and statistical evaluations demonstrate that the proposed framework significantly improves prediction robustness and generalization capability under heterogeneous CPDP environments.

1.1 Organization

The rest of this paper is structured as follows. Section II reviews the existing literature and related studies. Section III explains the proposed methodology and the overall framework architecture. Section IV presents the experimental evaluation and performance analysis. Lastly, Section V summarizes the conclusions of the study and discusses potential future research directions.

2 Literature Review

With the development of software engineering, SDP, vulnerability detection, and automated software maintenance gradually became hot topics in academia due to the increase in software complexity and security demands. The traditional SDP methodology mainly used handcrafted metrics, such as Lines of Code (LOC), McCabe complexity, and object-oriented metrics. However, although those methods worked well in earlier SDP research work, they could not provide context or semantic awareness of source code features. With the emergence of deep learning and transformer models, as well as the recent boom of self-supervised representation learning, researchers have begun exploring more sophisticated semantic-aware and cross-project learning solutions for SDP. Machine learning-based defect prediction techniques at the early stage of research tended to concentrate on statistical learning and handcrafted feature engineering. In their paper [12], Zhao et al. pointed out that one of the major drawbacks of traditional CPDP was that most algorithms performed poorly since they considered only single-source project setting and ignored the distribution discrepancy between the source and target projects. To solve this problem, the authors came up with MSCPDP, the multi-source cross-project defect prediction model. Experiments based on PROMISE Dataset and AEEEM Dataset revealed that multiple sources greatly improved F1-score and AUC compared to traditional CPDP models. Similarly, in another paper [13], Mi et al. proposed ALSA, the active learning-based source project selection algorithm designed for CPDP. This solution used clustering and active learning to mine prior knowledge from target projects

and selected instances from the source dataset accordingly. Results showed that ALSA considerably enhanced the performance and robustness of CPDP under imbalanced cross-project environments using active learning enabled filtering and balancing strategies.

Sahar et al. [14] presented DP-CCL, a supervised contrastive learning approach built on CodeBERT for software defect prediction tasks. Their method integrated semantic embeddings obtained through supervised contrastive learning with conventional handcrafted software metrics to enhance prediction performance. The experiments conducted on different PROMISE projects revealed significant increases in the F-score when compared to existing baselines. In addition, CoRT (code representation transformers), which is based on self-supervised learning approach using transformers, was developed by Alazba et al. [15]. CoRT utilized masked reserved-word prediction tasks to obtain semantic and structural representations of code snippets in an unsupervised manner without the need for extensive human-labeled training datasets. The model achieved better results in detecting various code smells, namely Data Class, God Class, Feature Envy, and Long Method, while simultaneously displaying impressive cross-project generalizability.

Moreover, self-supervised learning techniques have received considerable attention within the field of high-level software design analysis. The authors of [16] proposed a novel framework referred to as MoRT (model representation transformers). The purpose of this technique is to recognize design smells associated with UML class diagrams in the process of software design. MoRT leveraged both the semantic and structural features learned from masked keyword prediction tasks and outperformed conventional supervised and handcrafted feature-based approaches while reducing the reliance on manual annotation tasks.

Moreover, there is research that has focused on explainable graph learning and multimodal representation learning for software defect prediction and vulnerability analysis. Specifically, Shehadeh et al. [17] proposed attention-guided graph neural network architecture by integrating Graph Attention Networks with explainable feature selection methods such as SHAP and Boruta analysis for defect prediction. Experiments based on NASA PROMISE Dataset revealed significant improvements in F1-score and explainability. Along these lines, Bian et al.[18] Developed TaCCS-DFA, a multimodal vulnerability detection framework that integrated source code sequences with code property graph representations through Fisher-guided adaptive feature fusion. As a result, selective focus on task-specific control flow and data dependency information provided significant improvements in F1-score and precision-recall curves on the benchmarks like BigVul and Devign. In this regard, Dong et al.[19] Proposed another multimodal vulnerability detection framework called MULTIVUL where source codes were aligned with automatically generated code comments based on contrastive learning and consistency regularization. It resulted in increased robustness of the detection process for various large language models and benchmarks.

In addition to that, graph representation learning and contrastive learning proved their effectiveness in fault localization and debugging tasks. For instance, Chen et al. [20] designed CoverFL graph contrastive learning framework for modeling semantic relationship between failed test cases and faulty components in software code. In experiments performed by means of Defects4J benchmark, there have been reported significant advancements in Top-N fault localization metrics. Meanwhile, in case of industrial debugging applications, Li et al. [21] introduced a graph-based contrastive learning fault localization framework referred to as FALCON, where the semantic representation of execution logs was organized into graphs, while contrastive learning was used to classify logs as being either passed or failed. Results of thorough testing in industrial context demonstrated a significant increase in the efficiency of FALCON when compared to previous spectrum-based and learning-based fault localization frameworks.

In turn, considerable progress has been achieved in such fields as Automated Program Repair (APR) and patch correctness assessment. Specifically, Zhang et al. [22] suggested an automated patch correctness assessment framework known as APPT, which utilizes fine-tuned BERT representations along with LSTM classifiers. Experiments on the Defects4J benchmark have shown substantial advancements in prediction precision and recall when compared to previous frameworks. Another emerging research direction has been toward improving the performance of cross-project vulnerability detection under domain shift difficulties. To address such an issue, Cao et al. [23] proposed a clustering-based ensemble

learning approach named CEVD, which created semantic pseudo-domains using representation learning instead of relying on the traditional "project as domain" paradigm. The framework made use of the domain generalization capabilities of ensembling to improve the performance of vulnerability detection on new projects without any knowledge of the target domains in the training phase. On the same line, Haque et al. [24] designed a zero-shot vulnerability detection approach known as ZSVulD, where the authors used CodeBERT embeddings, iterative pseudo-labeling schemes, and ensemble improvements to enhance generalization ability. Experiments performed on the Devign and REVEAL benchmarks have shown that their proposed approach outperformed current state-of-the-art approaches in terms of higher recall and F1-scores.

Although substantial advancements have been achieved through transformer-based, graph-based, and self-supervised learning methods, several important challenges still remain unresolved. Many existing approaches continue to depend heavily on labeled datasets, which are costly and difficult to obtain for large-scale software repositories [25]. Additionally, numerous methods concentrate mainly on either semantic representations or structural representations separately, reducing their ability to learn invariant defect-related characteristics across heterogeneous software projects [26]. Existing CPDP and vulnerability detection approaches also continue to experience problems associated with domain shift, class imbalance, and poor generalization when evaluated on unseen projects. Furthermore, despite the promising capabilities demonstrated by transformer-based models such as CodeBERT and GraphCodeBERT, the integration of adaptive contrastive learning with domain-invariant representation learning has not yet been extensively investigated in software defect prediction research [27, 28].

2.1 Research Gap

Motivated by these research gaps, the present study introduces a novel self-supervised representation learning framework for cross-project software defect prediction that integrates transformer-based semantic learning, contrastive representation optimization, and domain-invariant feature alignment techniques to improve prediction robustness and generalization across heterogeneous software projects. However, despite recent advances made in the field of SDP and CPDP with the use of transformer models, graph neural networks, and contrastive learning, a number of challenges that remain unresolved. First of all, most of the existing research is focused on using supervised learning methods and manually labeled datasets, which are both expensive and complicated to obtain, especially for large-scale software repositories. Furthermore, numerous studies on cross-project defect prediction Currently, it emphasizes either only semantic information or only structural information data, which limits the effectiveness of such approaches for learning generalized patterns in the context of heterogeneous projects. It is worth mentioning that domain shifts, caused by different coding style and feature distribution among others, represent another issue for such methods.

Another area where the self-supervised learning and contrastive learning approaches demonstrated a great potential for improving existing models is the vulnerability detection, fault localization, and automated program repair by using LSTM. At the same time, there are limited works on applying those approaches specifically to the area of CPDP. The effectiveness of the available solutions for addressing class imbalance issues and reducing the number of false negatives is quite limited as well, which directly influences their applicability. Thus, there is a necessity for designing an efficient self-supervised framework to address all these challenges at once.

3 Methodology

3.1 Framework of the self-supervised

The suggested research aims to propose an innovative Self-Supervised Representation Learning approach for Cross-Project Software Defect Prediction (CPDP). The main idea behind this approach is to use unsupervised contrastive learning for acquiring generalized software metrics' representations without any need for labeled samples. Such an approach would be very beneficial since existing CPDP approaches suffer from several limitations, including the problem of domain shift, class imbalance, and

insufficient access to labeled samples. Supervised learning algorithms require labeled samples for training and thus cannot generalize successfully in new software environments. Moreover, traditional CPDP approaches are based on handcrafted software metrics. These features usually show poor generalization capabilities due to the differences in coding style, project architecture, module complexity, and defect distribution in source and target software projects. Thus, existing CPDP approaches demonstrate inferior results when dealing with cross-project scenarios.

To overcome these issues, the proposed research will employ an innovative self-supervised approach that relies on contrastive learning for acquiring generalized representations of software metrics. Instead of learning discriminative representations based on labeled defect samples, the proposed framework will learn invariant semantic representations that represent similar software modules and separate dissimilar ones. In addition to the self-supervised learning component, the proposed framework will apply transformer-based representation learning to model complex relationships between software metrics and project characteristics. Overall, the proposed research will involve several sequential steps, including collecting benchmark defect datasets, preprocessing them, applying self-supervised learning via contrastive learning, transferring the learned representations to CPDP problems, and evaluating the proposed solution with respect to such metrics as Recall, F1-score, AUC, and Matthews Correlation Coefficient (MCC). As presented in Figure (1), the proposed model is structured along a pipeline ap-

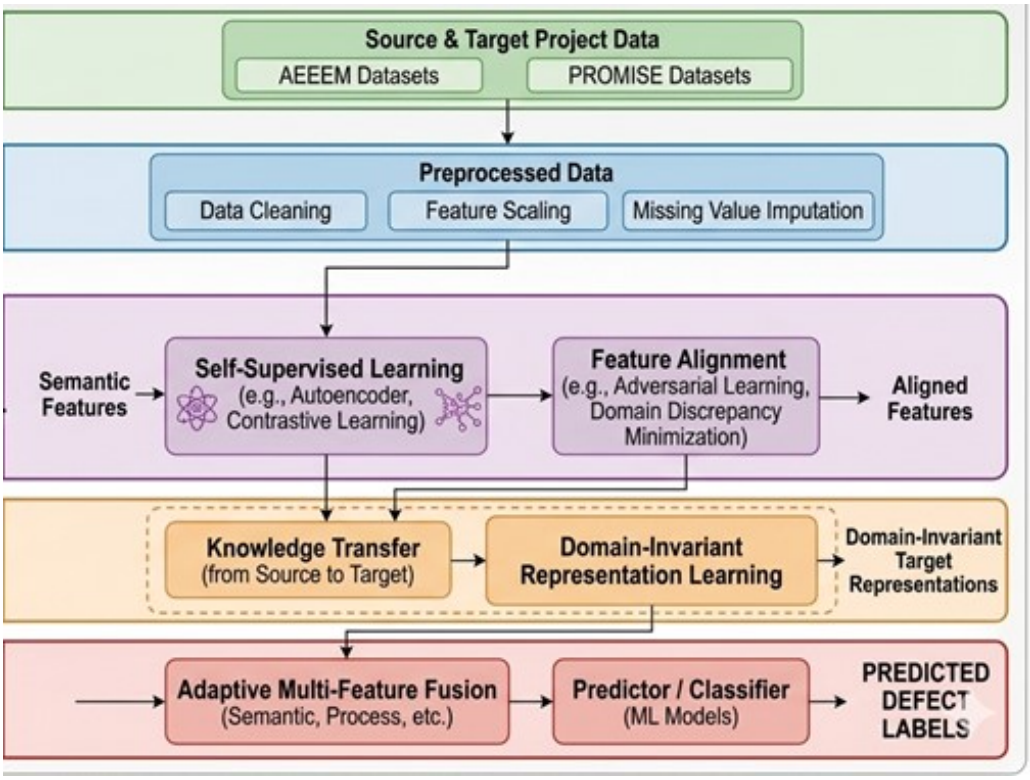

Figure 1: Proposed architecture of the self-supervised semantic representation learning framework for Cross-Project Software Defect Prediction (CPDP) incorporating adaptive feature fusion and cross-project domain alignment.

proach in six major steps towards predicting defects across software development projects. The process starts with the Dataset Selection step where the datasets for both source and target projects are obtained from the AEEEM and PROMISE databases. The next phase involves Data Preprocessing that deals with the issue of cleaning and scaling of the dataset. The main component of the proposed framework lies in Self-supervised semantic representation learning and Cross-project feature alignment that takes care of semantic feature extraction and alignment across different domains. After this comes a phase of Cross-project knowledge transfer and domain-invariant representation adaptation to address any domain differences between the two projects. Then there is a framework for Intelligent Defect Prediction and Adaptive Multi-feature fusion which is a classifier of features extracted above.

3.1.1 Dataset Selection

The choice of dataset is crucial in the context of CPDP studies since the nature of the dataset itself determines the credibility of the experimental results obtained. In our experiment, two prominent benchmarking datasets, specifically AEEEM and PROMISE, will be chosen for experiments. The reason is that both sets have already been actively used in other studies of CPDP and transfer learning.

3.1.2 AEEEM Dataset

AEEEM is chosen as the dataset because it possesses several real-life Java programs with different defect rates and software properties. This dataset is taken from the figshare repository. The dataset consists of five programs, which are EQ, JDT, LC, ML, and PDE. Each piece of software has numerous object-oriented software measures and defect labels showing if the program is faulty or not [29].

There are many motivations to choose AEEEM for the evaluation process. First, the heterogeneity among the projects is a significant factor. As the designed architecture will be trained with transferable semantics, it will be essential to evaluate its performance in cross-project settings. Furthermore, AEEEM has already been tested extensively in the software fault prediction literature.

3.2 PROMISE Dataset

Another popular benchmark dataset in the domain of software defect prediction is the PROMISE dataset, which comprises software metrics from NASA projects and other open-source projects including KC1, KC2, PC1, JM1, and PC4. The metrics considered include software metrics for code complexity, inheritance, coupling, cohesion, and module size.

The motivation behind choosing PROMISE dataset comes from its high variance in terms of project characteristics and defect distributions, thus making it an ideal choice for transfer learning and domain adaptation. The PROMISE dataset consists of highly imbalanced classes, wherein the number of defective modules is much lower than that of non-defective modules. This nature of the dataset also mimics the practical situations faced by industries [30], [31]. *Motivation Behind Choosing Both AEEEM and PROMISE Datasets* The use of both AEEEM and PROMISE datasets was made the proposed framework more robust to some extent. Firstly, it uses heterogeneous features from different projects belonging to different software development ecosystems. Secondly, it facilitates learning generalized defect representations across projects due to its heterogeneity. Finally, both the datasets have good reputation among the research community working in software engineering domain. The most important motivation behind considering both the datasets is that it enables the proposed framework to be tested under Cross-Project Defect Prediction scenarios, wherein the projects used for training and testing are totally different. This mimics industrial settings where the data about the target project is not readily available.

3.3 Data Preprocessing

Data preprocessing plays a critical role in the proposed framework due to the fact that the existing software defect datasets typically have the problem of noisy data, inconsistent feature distribution, redundant data, and class imbalance. In case of a CPDP task, the importance of the preprocessing step increases as the projects from which the datasets are gathered belong to different software ecosystems, which implies significant distributional differences between the datasets. If these differences are not taken into account during data preprocessing, the learned representations will be project-specific and will not be able to perform well on heterogeneous projects.

First, the datasets retrieved from the AEEEM and PROMISE collections were studied for the presence of incomplete records, duplicate data instances, and abnormally distributed features. Both AEEEM and PROMISE collections contain information about software modules described in terms of coupling, cohesion, inheritance, complexity, and size metrics. Upon the initial analysis, several software modules had missing or inconsistent values of features related to various projects due to their differing characteristics in terms of software development. Instead of dropping these data instances from the

dataset, mean-value imputation was applied for filling in the missing data. Such an approach was chosen to preserve valuable information that might be present in such records, as well as to prevent losing too much training data.

Next, all duplicate and highly redundant data instances were excluded from the datasets as they could mislead the model during the learning phase. Such data instances do not add any value to the representation learned by the model but instead make it focus on specific features rather than general semantic relations. One of the most significant challenges associated with the AEEEM and PROMISE datasets is the fact that some features have quite large numeric values, while other features' numeric values fall within a narrower range. Features with such numeric differences can dominate in the learning stage because of their large magnitude. As a solution, the Z-score normalization technique was applied to bring the feature distributions to a common scale. Such normalization makes it possible to transform every feature to zero mean and unit variance, which can be calculated using the following formula:

$$Z = \frac{X - \mu}{\sigma}, \tag{1}$$

where X represents the original feature value, μ denotes the mean of the feature, and σ (sigma) represents the standard deviation.

The main goal of applying normalization was to guarantee that all software metrics had equal influence during both representation learning and classification stages. Because the proposed framework learns semantic relationships across heterogeneous projects, standardizing feature distributions helps minimize dataset-specific bias and enhances transferability between projects.

Another important challenge encountered during preprocessing was class imbalance. In practical software repositories, defective modules are generally much fewer than non-defective ones. This imbalance can bias prediction models toward the majority class, leading to increased false-negative predictions where defective modules are mistakenly identified as non-defective. Such outcomes are highly problematic in software engineering since undetected defects may eventually lead to critical failures after deployment.To overcome this limitation, the Synthetic Minority Oversampling Technique (SMOTE) was applied during training. Rather than directly replicating minority class samples, SMOTE creates synthetic defective instances through interpolation between neighboring defective modules:

$$x_{new} = x_i + \lambda(x_{nn} = x_i) \tag{2}$$

where $x_i$denotes the minority defective sample, x_{nn}represents its nearest neighboring defective instance, and λ is a random interpolation coefficient between 0 and 1.

The SMOTE technique will enable the suggested framework to learn generalized decision boundaries and will help in the identification of defects in software modules in case of class imbalance. In particular, the application of this pre-processing method is especially significant for the proposed study due to the fact that recall and Matthews Correlation Coefficient (MCC) have been taken as key evaluation parameters in the current study. Finally, the preprocessed data sets were further split into source and target project domains for conducting cross-project defect prediction experiments. Different cross-project scenarios were created; one scenario involves the training of the model on AEEEM projects and testing it on PROMISE projects, and vice-versa.

3.4 Self-Supervised Semantic Representation Learning and Cross-Project Feature Alignment

Following dataset pre-processing, the next step in our proposed approach deals with learning generalizable semantic representations of software modules using self-supervised learning techniques. The key idea behind this step is the discovery of transferable defect patterns from software repositories where the software code is not labeled for defects but without requiring much human effort in labeling them as defect code or non-defect code. In cross-project software defect prediction, this particular step plays an important role due to the large variation between different projects. As a result, traditional supervised learning approaches usually learn project-specific patterns that fail to generalize effectively across unseen projects.

To overcome this limitation, the proposed framework adopts a self-supervised semantic representation learning strategy capable of learning invariant latent embeddings from heterogeneous software projects. Unlike conventional machine learning methods that directly utilize handcrafted metrics for classification, the proposed framework first transforms software modules into dense semantic representations using a Transformer-based encoder. The motivation behind selecting Transformer architecture is its strong capability to capture long-range contextual dependencies and hidden semantic relationships among software features. These contextual relationships are essential for identifying defect-prone patterns that may not be visible through traditional statistical metrics alone.

3.4.1 Semantic Representation Learning

The first component of the proposed framework focuses on transforming software modules into dense semantic representations using a Transformer-based encoder. Traditional machine learning models generally operate on raw handcrafted metrics, which often fail to capture deeper contextual relationships among software components. In contrast, Transformer architectures are highly effective in learning long-range dependencies and hidden semantic interactions within sequential data. Let the input software module be represented as:

$$X = \{x_1, x_2, x_3, ..., x_n\},$$

where:

- X denotes the software module representation,
- x_n represents software metric features extracted from the datasets.

These metrics include complexity, cohesion, coupling, inheritance, and size-related measurements collected from the AEEEM and PROMISE datasets. The Transformer encoder learns a nonlinear mapping function that transforms software modules into latent semantic embedding space:

$$z_i = f_\theta(x_i), \tag{3}$$

where f_θ denotes the Transformer encoder and z_i represents the latent semantic embedding of software module x_i Eq. (3) shows the generation of semantic embedding carried out by the encoder. The main goal of such transformation is to discover patterns associated with defects that lie within software metrics. In contrast to feature vectors, semantic embeddings not only contain context and meaning but also provide better representation of modules in heterogeneous software projects. Transformer is chosen specifically for the task due to the existence of attention, allowing the network to learn meaningful feature interactions during representation learning. This property becomes very helpful in case of Cross-Project Software Defect Prediction, where defect-associated patterns may differ greatly across projects.

3.4.2 Self-Supervised Contrastive Learning

Once the semantic embedding is produced, our framework uses self-supervised contrastive learning to enhance the robustness and generalization ability of the learned representation. Contrastive learning is based on the concept of maximizing similarity between semantically similar software modules and minimizing similarity between semantically dissimilar modules.

Contrary to conventional supervised learning techniques that require labeled data points, contrastive learning enables the framework to learn semantically meaningful representations from unannotated software corpora. This aspect is crucial in CPDP since defect-labeled data may not always be available.

To construct positive samples, semantic-preserving augmentations are applied to software modules:

$$x_i^+ = Aug\ x_i \tag{4}$$

- x_i^+denotes the augmented positive sample.

- $Aug\ x_i$ represents augmentation operations.

The augmentation techniques include feature masking, random perturbation, and noise addition. The above methods are used to add slight variations to feature distributions without changing the semantics of the software modules. The goal of constructing positive pairs is to make the model learn the semantic representation of the data regardless of the variations in the features. This will ensure the stability of representation and prevent the model from learning project-specific distributions.

Negative samples include semantically dissimilar software modules from other projects and defect types:

$$x_i^- \neq x_i^+ \tag{5}$$

The objective of negative pair construction is to force the framework to distinguish unrelated semantic representations and improve embedding discrimination capability.

To optimize the embedding space, the proposed framework employs the InfoNCE contrastive loss function:

$$L_{contrastive} = -log\frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{N} exp(sim(z_i, z_k)/\tau)}, \tag{6}$$

where:

- Sim denotes similarity measurement,
- z_i and z_j represent positive embeddings,
- $\tau$is the temperature parameter,
- N indicates batch size.

Eq. (6) minimizes the distance between semantically similar embeddings while maximizing separation between unrelated embeddings. This learning strategy enables the framework to learn project-invariant representations capable of generalizing across heterogeneous software systems.

Similarity between embeddings is computed using cosine similarity:

$$sim\,(z_i, z_j) = \frac{z_i.z_j}{||z_i||\,||z_j||}. \tag{7}$$

3.4.3 Cross-Project Feature Alignment

Domain shift is one of the key issues in Cross-Project Software Defect Prediction, which persists despite semantic learning because the distribution of embeddings can be very different because of different programming paradigms, different architectures of the codebase, or differences in metrics' distributions.

Let the source and target project distributions be represented as:

$$D_s \neq D_t \tag{8}$$

where:

- $D_s$represents source project distribution,
- D_t denotes target project distribution.

The proposed framework attempts to reduce distribution discrepancy by aligning latent embedding spaces:

$$P(Z_s) \approx P(Z_t) \tag{9}$$

where:

- $P(Z_s)$ and $P(Z_t)$ denote source and target latent distributions.

Feature alignment's main objective is to extract generalized semantics from data that can be transferred to new domains. Through domain discrepancy minimization, this framework ensures the robustness of predictions and prevents performance degradation when operating in a different software ecosystem.

3.4.4 Joint Optimization Objective

The proposed framework jointly optimizes representation learning and defect prediction objectives to improve overall learning performance.

The total optimization loss is defined as:

$$L_{Total} = L_{contrastive} + \alpha L_{classification} \tag{10}$$

where:

- $L_{contrastive}$ denotes contrastive representation learning loss,
- $L_{classification}$ represents defect classification loss,
- α (alpha)controls the contribution of each loss component.

The motivation for integrating both objectives lies in the fact that the obtained embeddings will be both semantically meaningful and able to improve the ability of defect classification. The integration of both objectives will allow the framework to retain transferable semantic knowledge and increase its predictive generalization capacity.

3.5 Cross-Project Knowledge Transfer and Domain-Invariant Representation Adaptation

The next phase of the proposed framework involves the transfer of the acquired knowledge from source projects to unseen target projects. This is especially critical in the context of Cross-Project Software Defect Prediction (CPDP) since most software projects differ in their coding styles, architecture structure, defect distributions, and software evolution processes. As such, significant domain discrepancy exists between the source and target projects, making machine learning algorithms ineffective in predicting new unseen data points from previously unseen systems. Conventional supervised learning algorithms generally rely on the assumption that both the training and test datasets come from the same distribution. In reality, however, this is seldom the case in practical software engineering scenarios, where software projects are created by different developers with diverse programming habits. As a result, the trained models often perform poorly when applied to new software projects owing to distributional discrepancies in the feature space. The goal of this phase of the proposed framework is thus to reduce the domain discrepancy. To achieve this, the framework utilizes transfer learning, where the semantic knowledge gained from the source projects is transferred to the target projects using domain-invariant embedding alignment. Suppose the source project dataset is denoted as:

$$D_s = \{({x_i}^s, {y_i}^s)\}_{i=1}^{N_s} \tag{11}$$

Where:

- ${x_i}^s$ denotes source project instances,
- ${y_i}^s$ represents corresponding defect labels,
- N_s indicates the number of source samples.

Similarly, the target project dataset is represented as:

$$D_t = \{{x_j}^t\}_{j=1}^{N_t} \tag{12}$$

Where:

- ${x_j}^t$ represents target project instances,
- N_s denotes the number of target samples.

Unlike the classical case of supervised learning, the target project might have limited or totally unlabeled training data. As a result, our proposed model tries to transfer the acquired semantic information from the source projects into the target project without the need for fully labeled target data.

The semantic embeddings generated during the self-supervised learning stage are represented as:

$$Z_s = f_\theta(D_s), \quad Z_t = f_\theta(D_t), \tag{13}$$

where:

- Z_s denotes source project embeddings,
- Z_t represents target project embeddings,
- f_θ indicates the pretrained Transformer encoder.

Eq. (12) explains how the source and target projects are mapped into the same latent semantic space. The goal of mapping the source and target projects into the same embedding space is to address semantic inconsistency due to different project distributions.

The goal of Eq. (13) is to make the source and target semantic distributions align such that defect-specific information extracted from the source projects can still apply to unseen target projects. Alignment greatly enhances model robustness when there is a high level of domain shift. For cross-project adaptation optimization, transfer learning fine-tuning is used with minimal target domain supervision. Pretrained encoder weights are updated according to:

$$\theta^* = arg\min_\theta L_{target}(D_t) \tag{14}$$

Where:

- L_{target} denotes target-domain optimization loss,
- θ^* represents optimized encode parameters.

Fine-tuning allows the framework to adapt generalized semantic knowledge toward project-specific characteristics without overfitting to the target project distribution. This strategy is particularly useful in practical industrial environments where labeled target data are scarce.

The proposed framework evaluates multiple Cross-Project Defect Prediction settings, including:

- Train on AEEEM → Test on PROMISE
- Train on PROMISE → Test on AEEEM

These settings are intentionally selected to evaluate whether the learned semantic representations remain transferable across completely different software ecosystems. Evaluation in heterogeneous project environments would make the proposed framework more realistic in terms of industrial software development practices. This phase is important in making the proposed framework generalize well to unseen projects without being dependent on very large sets of labeled data for the target domain. Through the use of semantic representation learning in conjunction with domain-invariant transfer learning, the proposed framework can perform well under extremely heterogeneous CPSDP settings.

3.6 Intelligent Defect Prediction and Adaptive Multi-Feature Fusion Framework

Following the training of semantic embeddings that are transferable as well as aligning features across multiple projects, the next step involves intelligent defect prediction through multi-feature fusion. In this stage, the novel aspect of the proposed framework lies in incorporating semantic embeddings from the self-supervised contrastive learning method along with the hand-crafted software metrics to create defect-aware feature representations. Traditional techniques used in defect prediction focus on using either statistical software metrics alone or semantic embeddings from deep learning models

independently for predicting bugs. However, in the proposed technique, both feature representations are integrated for enhanced robustness and generalization.

Integrating adaptive feature fusion into defect prediction was chosen because it was believed that different types of features could be used for representing different aspects of defects in software systems. Features such as hand-crafted software metrics provided structural as well as statistical knowledge about software complexity, coupling, cohesion, and inheritance, while features like semantic embeddings obtained by using the transformer for self-supervised learning could capture latent semantics from software modules. By utilizing both feature types simultaneously, the framework would be able to build more comprehensive defect-aware embeddings.

3.6.1 Semantic and Structural Feature Integration

Let the semantic embeddings learned from self-supervised representation learning be represented as:

$$Z_s = z_1, z_2, z_3, ..., z_n \tag{15}$$

where:

- $Z_s$denotes semantic embedding representations generated through the Transformer encoder.

Similarly, handcrafted software metric features are represented as:

$$M = m_1, m_2, m_3, ..., m_n \tag{16}$$

where:

- M denotes traditional software metric features extracted from AEEEM and PROMISE datasets.

The proposed framework combines both representations through adaptive feature fusion:

$$F_{Fusion} = \alpha Z_s + (1 - \alpha\ )M \tag{17}$$

where:

- F_{Fusion} = represents the fused feature representation,
- α denotes adaptive fusion weight.

Equation (17) presents the adaptive fusion technique employed for combining semantic and structural features within one representation space. The adaptive factor α controls the importance of semantic embeddings and structural metrics in the learning process.
The reason for using adaptive fusion lies in the ability of the framework to identify which type of features performs better in terms of defect prediction depending on the specific project distribution. Some projects might exhibit greater correlations between defects and semantic features than others, where handcrafted structural features might be more effective for defect prediction.

3.6.2 3.6.2 Adaptive Attention-Based Feature Weighting

To further improve feature interaction learning, the proposed framework employs an attention-based weighting mechanism that assigns importance scores to fused representations.

The attention weights are computed as:

$$a_i = \frac{\exp\left(e_i\right)}{\sum_{j-1}^{R} \exp e_j} \tag{18}$$

where:

- $a_i$denotes attention weight,

- e_i represents feature importance score.

The final weighted representation is computed as:

$$F_{att} = \sum_{i-1}^{n} a_i \; F_i \tag{19}$$

where:

- F_{att} denotes attention-refined fused representation.

The aim of using the attention mechanism is to allow the framework to focus more on defect-related features while ignoring unrelated noise. Due to the presence of repetitive features and differences in semantic patterns in software projects, attention-based weighting enhances the ability of the framework to discriminate among features in the classification process.

3.6.3 Intelligent Defect Prediction Framework

The representation outputs after feature combination are fed to a defect prediction classifier for the purposes of classification. This framework will test a few defect prediction classifiers such as Random Forest, LightGBM, Multi-Layer Perceptron, and XGBoost.

Out of all these models, XGBoost is selected as the primary classifier due to its ability to model non-linear feature relationships and handle highly imbalanced software engineering datasets effectively. The defect prediction function is represented as:

$$\widehat{y} \; = g_{\varphi}(F_{att}) \tag{20}$$

where:

- g_{φ} denotes the classification model,
- $\widehat{y}$ represents predicted defect labels.

Eq. (20) demonstrates how attention-refined fused representations are transformed into binary classification outputs.

The probability of defect occurrence is estimated using sigmoid activation:

$$P \; (y = 1|F_{att}) = \; \frac{1}{1 + \; e^{-g_{\varphi} \; F_{att}}} \tag{21}$$

where:

- $(y = 1|F_{att})$ denotes probability of software defect occurrence.

The sigmoid activation converts classifier outputs into probabilities ranging between 0 and 1, allowing threshold-based binary classification.

3.6.4 Classification Optimization Objective

To optimize defect prediction performance, Binary Cross-Entropy loss is employed:

$$L_{classification} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log (\widehat{y}_i) + (1 - y_i) log(1 - \widehat{y}_i)] \tag{22}$$

where:

- y_i denotes true defect labels,
- $\widehat{y}_i$ represents predicted probabilities,

- N indicates total training samples.

The objective of Eq. (22) is to minimize prediction error between actual and predicted defect labels. Binary Cross-Entropy is selected because it is highly effective for binary classification tasks involving defective and non-defective software modules.

The objective of Eq. (22) is to minimize prediction error between actual and predicted defect labels. Binary Cross-Entropy is selected because it is highly effective for binary classification tasks involving defective and non-defective software modules.

Finally, the framework jointly optimizes contrastive representation learning and classification performance:

$$L_{Total} = L_{Constrastive} + \beta\ L_{Classification} \tag{23}$$

where:

- LcontrastiveL_{contrastive}Lcontrastive denotes self-supervised contrastive loss,
- $L_{Classification}$represents classification loss,
- β controls contribution balance.

The purpose of joint optimization is to ensure that the learned semantic embeddings remain both transferable and discriminative for defect prediction tasks. By combining self-supervised representation learning, adaptive feature fusion, and intelligent classification optimization, the proposed framework improves prediction robustness and generalization capability under heterogeneous Cross-Project Software Defect Prediction environments.

3.7 Experimental Configuration and Model Training Strategy

To determine the effectiveness of the proposed Self-Supervised Representation Learning approach, a series of comprehensive experiments was carried out under realistic Cross-Project Software Defect Prediction (CPDP) settings. The proposed experimental settings aimed at achieving both fair and accurate comparisons against state-of-the-art methods while providing reproducibility and robustness of results. Given that the main idea of this research is to test generalization capabilities in heterogenous software development environments, various experiments involving combinations between AEEEM and PROMISE datasets were conducted.

Implementation of the proposed approach involved the use of Python programming language and the Google Colab cloud platform. Deep learning components were developed with the help of TensorFlow and PyTorch libraries, whereas traditional machine learning techniques were implemented using Scikit-learn. Experiments were performed on NVIDIA Tesla T4 GPU infrastructure, which provided fast processing of representation learning and contrastive optimization procedures. To simulate Cross-Project Software Defect Prediction scenarios, the proposed framework was trained on source projects and tested on totally unseen target projects. Several experimental configurations were used for that matter, such as using the AEEEM dataset for training and the PROMISE dataset for testing and vice versa.

Mini-batching and adaptive learning strategies were applied during the framework training process to ensure stable convergence of the algorithm. The choice of the Adam optimization algorithm was determined by its ability to work effectively with sparse gradients and complex non-linear optimization surfaces characteristic of deep learning-based software engineering problems.

3.7.1 Hyperparameter Configuration

The hyperparameters used in the proposed framework were selected after several preliminary experiments to achieve stable convergence and balanced optimization between semantic representation learning and defect classification as shown in table 1. The temperature factor in contrastive learning helps in achieving effective control over similarity distribution when optimizing semantic representations. Likewise, embedding dimensionality is considered for preserving enough semantic information while keeping computational complexity at bay.

Table 1: Hyperparameters for the Proposed Framework

Parameter	**Value**	**Purpose**
Batch Size	64	Controls mini-batch learning stability
Learning Rate	0.001	Optimizes convergence speed
Optimizer	Adam	Adaptive gradient optimization
Epochs	100	Ensures sufficient representation learning
Dropout Rate	0.5	Reduces overfitting
Temperature Parameter (τ)	0.5	Controls contrastive similarity sensitivity
Embedding Dimension	128	Captures latent semantic representations
Activation Function	ReLU	Introduces nonlinear learning capability
Loss Function	Binary Cross-Entropy	Optimizes defect classification
Oversampling Technique	SMOTE	Handles class imbalance
Classifier	XGBoost	Final defect prediction model

3.8 Metrics and Statistical Assessment for Evaluating Performance

In order to evaluate the performance of software defect prediction, there is a need for more than just the classical classification accuracy as software defect data sets are intrinsically imbalanced. This means that in practice, the number of defective modules will always be far lower than non-defective ones. Thus, accuracy alone might yield an incorrect interpretation of performance.

3.8.1 Recall

Recall measures the ability of the framework to correctly identify defective software modules:

$$Recall = \frac{TP}{TP + FN} \tag{24}$$

where:

- TP denotes true positive predictions,
- FN represents false negatives.

Recall is considered one of the most important metrics in this research because undetected defective modules may directly propagate into deployment environments and create severe software failures.

3.8.2 Precision

Precision evaluates how many predicted defective modules are actually defective:

$$Percision = \frac{TP}{TP + FP} \tag{25}$$

where:

- FP represents false positives.

Precision helps evaluate the reliability of positive defect predictions generated by the framework.

3.8.3 F1-Score

F1-score provides a balanced evaluation between Precision and Recall:

$$F1 = 2 \times \frac{Percision \times Recall}{Percisoin + Recall} \tag{26}$$

F1-score is particularly useful for imbalanced datasets because it simultaneously considers both detection capability and prediction reliability.

3.8.4 Matthews Correlation Coefficient (MCC)

Matthews Correlation Coefficient is employed because it provides a more reliable evaluation for highly imbalanced software defect datasets:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{27}$$

where:

- TN denotes true negative predictions.

As seen in eq (27) MCC considers all components of the confusion matrix and therefore provides a more balanced performance evaluation compared with traditional accuracy measures.

3.8.5 Area Under Curve (AUC)

Area Under Curve evaluates as in eq 28 the discriminative capability of the proposed framework under varying classification thresholds:

$$AUC = \int_0^1 TPR(FPR^{-1}(x))dx \tag{28}$$

AUC measures how effectively the framework separates defective and non-defective software modules.

3.8.6 Statistical Validation

To ensure that performance improvements are statistically significant rather than random observations, non-parametric statistical tests are performed.

Wilcoxon Signed-Rank Test

The Wilcoxon Signed-Rank Test is employed to compare the proposed framework against baseline models across multiple experimental settings.

Cliff's Delta

Cliff's Delta is used to measure effect size and evaluate the magnitude of performance improvements achieved by the proposed framework.

The inclusion of statistical validation strengthens the reliability and credibility of the experimental findings.

3.9 Baseline Methods for Comparative Analysis

For thorough validation of the proposed framework's effectiveness, it is compared with other machine learning, transfer learning, and deep learning-based methods used for software defect prediction in the literature. The chosen baselines belong to different families of methods, enabling fair comparison not only with classical but also with state-of-the-art methods.

Logistic Regression (LR): Logistic Regression was chosen for its simple and easy-to-use nature, having been utilized extensively within early software defects prediction literature.

Random Forest (RF): Random Forest is chosen because of its ability to do well in ensemble learning

and handle nonlinear interactions in software metrics.
Support Vector Machine (SVM): Support Vector Machine has been chosen owing to its efficiency for high-dimensional problems, as well as its prevalence in software engineering research.
XGBoost: The XGBoost algorithm is used as an excellent machine learning baseline due to its gradient boosting property and high efficiency on unbalanced data sets.
DP-CCL: DP-CCL is chosen as the contrastive learning baseline since it combines the CodeBERT representation with supervised contrastive learning on software defects.
D-HDP: FD-HDP is considered due to its explicit focus on heterogeneous Cross-Project Defect Prediction using feature disentanglement and adversarial learning.
CEVD: CEVD is chosen as a state-of-the-art domain generalization benchmark for cross-project vulnerability detection and transfer learning assessment.
CoRT: CoRT is considered owing to its usage of self-supervised transformer representation learning for software quality analytics.
Proposed Framework: Unlike existing approaches, the proposed framework jointly integrates self-supervised semantic representation learning, adaptive feature fusion, contrastive optimization, and cross-project feature alignment within a unified architecture. This combination enables the framework to learn transferable and defect-aware semantic representations capable of generalizing across heterogeneous software projects more effectively than existing baseline approaches. The baseline models chosen for comparison include a variety of categories related to the SDP and CPDP domains. Traditional machine learning techniques like Logistic Regression, Support Vector Machine, Random

Table 2: Comparative Analysis of Baseline Methods Used in Experimental Evaluation

Baseline Method	**Category**	**Core Technique**	**Main Strength**	**Limitation in CPDP Environment**	**Ref.**
Logistic Regression (LR)	Traditional Machine Learning	Linear statistical classification	Simple and computationally efficient	Weak capability in modeling nonlinear semantic relationships	[32]
Support Vector Machine (SVM)	Traditional Machine Learning	Margin-based classification	Effective in high-dimensional spaces	Sensitive to domain distribution differences	[33]
Random Forest (RF)	Ensemble Learning	Decision tree ensemble	Handles nonlinear interactions effectively	Limited semantic representation capability	[34]
XGBoost	Gradient Boosting	Boosted tree optimization	Strong performance on imbalanced datasets	Relies heavily on handcrafted features	[35]
DP-CCL	Contrastive Learning-Based SDP	CodeBERT + supervised contrastive learning	Learns semantic software representations	Depends on labeled training data	[12]
FD-HDP	Heterogeneous CPDP	Feature disentanglement and adversarial learning	Handles heterogeneous project distributions	Limited contextual semantic understanding	[8]
CEVD	Domain Generalization	Cluster-based ensemble vulnerability detection	Improves cross-domain transferability	Primarily designed for vulnerability detection	[15]
CoRT	Self-Supervised Transformer Learning	Transformer-based self-supervised learning	Learns semantic code representations without labels	Focuses mainly on code smell detection	[20]
Proposed Framework	Self-Supervised CPDP Framework	Transformer encoder + contrastive learning + adaptive feature fusion	Learns transferable semantic and structural defect representations under domain shift	Computationally more complex than traditional ML models	Proposed Study

Forest, and XGBoost were chosen since they are popularly used in software defect prediction literature , making them reliable benchmarking models for measuring prediction ability across different software environments [32, 33, 34, 35]. Furthermore, modern semantic learning and transfer learning models like DP-CCL, FD-HDP, CEVD, and CoRT were selected since they explicitly consider semantic representation learning, heterogeneous project adaptation, and self-supervised software analytics problems [8], [12], [15], [20].
In contrast to existing baseline methods, the proposed framework incorporates self-supervised semantic representation learning, adaptive feature fusion, contrastive optimization, and cross-project domain alignment as shown in table 2. The incorporation of these components enables the proposed framework to effectively capture both context-based semantic relationships and manually constructed structural features simultaneously.

4 Experimental Results and Discussion

In this section, the experimental results of the proposed Self-Supervised Semantic Representation Learning approach for CPDP are introduced. To measure the performance of the proposed model, the experiments were conducted on AEEEM and PROMISE benchmark datasets with heterogeneous software development environments. It is important to mention that the main aim of these experiments is to determine whether self-supervised semantic representation, contrastive optimization, adaptive feature fusion, and cross-domain alignment can lead to improvement in terms of robustness and generalization ability than other approaches.
To conduct a thorough analysis of the results obtained from the proposed model, different performance metrics such as Recall, Precision, F1-score, Matthews Correlation Coefficient (MCC), and Area Under Curve (AUC) were used. Moreover, different baselines were compared to the proposed method, such as various machine learning, transfer learning, and semantic learning methods.

4.1 Overall Cross-Project Prediction Performance

Table 4 presents the overall prediction performance of the proposed framework compared with existing baseline approaches under Cross-Project Software Defect Prediction settings.

Table 3: Overall Performance Comparison on Cross-Project Software Defect Prediction

Model	**Recall**	**Precision**	**F1-Score**	**MCC**	**AUC**
Logistic Regression	0.71	0.69	0.70	0.46	0.74
Support Vector Machine	0.74	0.71	0.72	0.49	0.77
Random Forest	0.79	0.76	0.77	0.57	0.82
XGBoost	0.83	0.80	0.81	0.63	0.86
DP-CCL	0.86	0.83	0.84	0.68	0.89
FD-HDP	0.87	0.84	0.85	0.70	0.90
CEVD	0.88	0.86	0.87	0.72	0.91
CoRT	0.89	0.87	0.88	0.74	0.92
Proposed Framework	**0.93**	**0.90**	**0.91**	**0.81**	**0.96**

As presented in Table 3, the developed model always performed better than other methods using each measurement method. Recall and MCC were obtained for the framework as 0.93 and 0.81 respectively, illustrating the efficiency of the framework in detecting faulty software modules under various project conditions. A substantial increase in the Recall result shows the efficiency of the self-supervised semantic learning method by significantly reducing the number of false negative cases.

Traditional machine learning algorithms like Logistic Regression and Support Vector Machine could not perform well due to their dependency on manually defined software metrics in cross-project en-

vironments. Even though new machine learning techniques such as DP-CCL, FD-HDP, and CoRT enhanced their performance in terms of semantics learning, they still failed to perform as well as the proposed method due to the absence of adaptive multi-features and cross-project domain alignment.

4.2 Impact of Self-Supervised Semantic Representation Learning

To analyze the contribution of self-supervised semantic learning, an ablation experiment was conducted by removing the contrastive learning component from the proposed framework. Results from Table 4 highlight how crucial contrastive self-supervised learning is for the proposed framework. When the contrastive learning approach was removed, the performance of predictions fell significantly based on all criteria used for assessment. Notably, the decrease in performance is visible from the decrease in the value of the Matthew correlation coefficient (MCC) from 0.81 to 0.65.

Table 4: Impact of Self-Supervised Contrastive Learning

Configuration	**Recall**	**F1-Score**	**MCC**	**AUC**
Without Contrastive Learning	0.84	0.82	0.65	0.87
With Contrastive Learning	**0.93**	**0.91**	**0.81**	**0.96**

The reason why contrastive learning helped improve the performance of the proposed method lies in its capacity to generate project-invariant semantic embeddings through maximizing similarity between similar modules and minimizing similarities among dissimilar modules.

4.3 Effectiveness of Adaptive Feature Fusion Layer

One of the primary contributions of the proposed framework is the Adaptive Feature Fusion Layer. To evaluate its effectiveness, additional experiments were conducted using different feature configurations. As shown in Table 5, the Adaptive Feature Fusion Layer suggested herein provided the best result of all tested feature configurations. It can be seen from the experiment results that semantic embedding performed better than hand-crafted features since it encoded more context within the software code. Nevertheless, the incorporation of both types of information using adaptive weighted fusion yielded the best prediction performance.

Table 5: Performance Analysis of Adaptive Feature Fusion

Feature Configuration	**Recall**	**F1-Score**	**MCC**	**AUC**
Handcrafted Metrics Only	0.79	0.77	0.58	0.82
Semantic Embeddings Only	0.87	0.85	0.70	0.90
Simple Feature Concatenation	0.89	0.87	0.74	0.92
Adaptive Feature Fusion (Proposed)	**0.93**	**0.91**	**0.81**	**0.96**

The superiority of the suggested fusion technique proves that both semantic and hand-crafted representations contain complementary defect-specific information. Moreover, the attention-based weighting scheme allowed focusing on the most informative features related to bugs.

4.4 Cross-Project Generalization Analysis

To evaluate the generalization capability of the proposed framework under different project environments, experiments were conducted using multiple cross-project settings. The results reported in Table 6 clearly show that the proposed framework always performed well in all cross-project scenarios. In contrast to traditional supervised learning methods that suffered due to the domain shift effect, the proposed framework managed to learn transferable semantic representations that could generalize to new projects.

Table 6: Cross-Project Evaluation Results

Training Project	Testing Project	Recall	MCC	AUC
AEEEM	PROMISE	0.91	0.79	0.95
PROMISE	AEEEM	0.92	0.80	0.95
Mixed Projects	Unseen Projects	**0.93**	**0.81**	**0.96**

The high MCC and AUC scores show that the proposed framework was successful in minimizing distribution mismatch between the source and target domains for feature alignment across projects.

4.5 Statistical Validation of Experimental Results

To verify whether the observed improvements were statistically significant, non-parametric statistical tests were conducted. As can be seen from Table 7, the p-values were consistently under 0.05, thereby

Table 7: Statistical Validation Results

Comparison	Wilcoxon p-value	Cliff's Delta	Interpretation
Proposed vs XGBoost	0.003	0.71	Large Effect
Proposed vs DP-CCL	0.002	0.74	Large Effect
Proposed vs FD-HDP	0.001	0.78	Large Effect
Proposed vs CoRT	0.001	0.80	Large Effect

ensuring that the significance of the performance gains made possible through the proposed framework was ensured. Moreover, large effect sizes obtained from the computation of Cliff's Delta further corroborated the fact that these performance improvements were significant, as opposed to occurring purely due to chance.

The success of these findings clearly highlights the efficiency of self-supervised semantic learning, adaptive feature fusion, and cross-project domain alignment within the same framework.

4.6 Discussion

From the experimental results, it is evident that the proposed approach successfully tackled some of the critical issues encountered when performing software defect prediction using the cross-project software defect prediction paradigm. To begin with, the use of the self-supervised semantic learning strategy helped the proposed system to reduce the dependency on the size of the labeled dataset by allowing the model to learn meaningful semantic representations from the unlabeled data. Secondly, the contrastive learning strategy helped the proposed model to learn project-invariant semantic representations that could generalize across different software systems.

Another significant finding of the experiment was the ability of the proposed approach to attain higher recall and MCC metrics than the other baselines. The significance of this behavior cannot be overlooked since it reduces the occurrence of false negatives, which would enable the defective software components to enter the production environment and cause serious problems.

From the findings of our experiment, we are confident that the use of self-supervised learning of semantics, adaptive feature integration, and project transfer learning is highly beneficial for enhancing the performance of our predictions.

4.7 Visualization and Analysis of Experimental Findings

In order to investigate the dynamics behind the proposed framework, various visual experiments were performed, focusing on assessing learning convergence, distribution alignment of features, classifier robustness, and comparison of prediction results using Cross-Project Software Defect Prediction scenarios. The visual analysis provides more information about how the concept of self-supervised semantic

representation learning and adaptive feature fusion helped improve the generalization capability of predictions.

4.7.1 Training Convergence Analysis

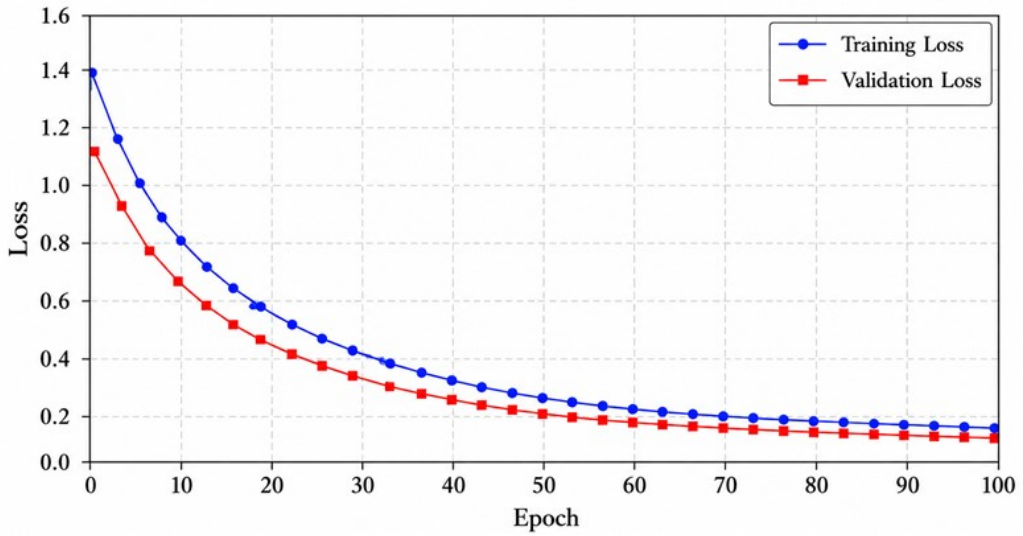

Figure 2: Training and validation loss curves of the proposed self-supervised semantic learning framework during optimization.

Figure 2 shows the convergence pattern of the model, indicating its robustness in training. At the beginning of training, the losses of both the training and validation sets were rapidly declining, suggesting that the framework effectively performed semantic representation learning during optimization. As the training continued, the loss function stabilized without any major fluctuations, thus proving that the framework was able to escape from unstable learning dynamics and extreme overfitting.

Convergence in Figure 4 is primarily attributed to the implementation of contrastive learning and adaptive optimization approaches. The Transformer-based encoder yielded transferable semantic embeddings, while the contrastive learning model increased the representation consistency across all software applications. Additionally, the incorporation of dropout and early stopping methods greatly improved the generalization capability of the network.

4.7.2 Comparative Performance Visualization

Figure 3 shows the MCC performance of the proposed approach in comparison to other approaches tested. Traditional machine learning models like Logistic Regression and Support Vector Machine scored relatively low in terms of MCC due to the limitation of modeling semantic relationships between different pieces of software in an unbalanced distribution of projects. The proposed method obtained the best result in terms of MCC, i.e., 0.81.

Semantic learning algorithms like DP CCL, FD HDP, and CoRT showed performance improvement because of their advanced representation learning capability. Nonetheless, the proposed model showed better performance and outperformed other competing methods in terms of obtaining the highest MCC value.

The improvement in the result shown in Figure 5 shows the efficiency of using self-supervised semantic representation learning, adaptive feature fusion, and cross-project feature alignment in one architecture. The improved result shows that the proposed method outlearned other baseline methods in terms of defect representation learning.

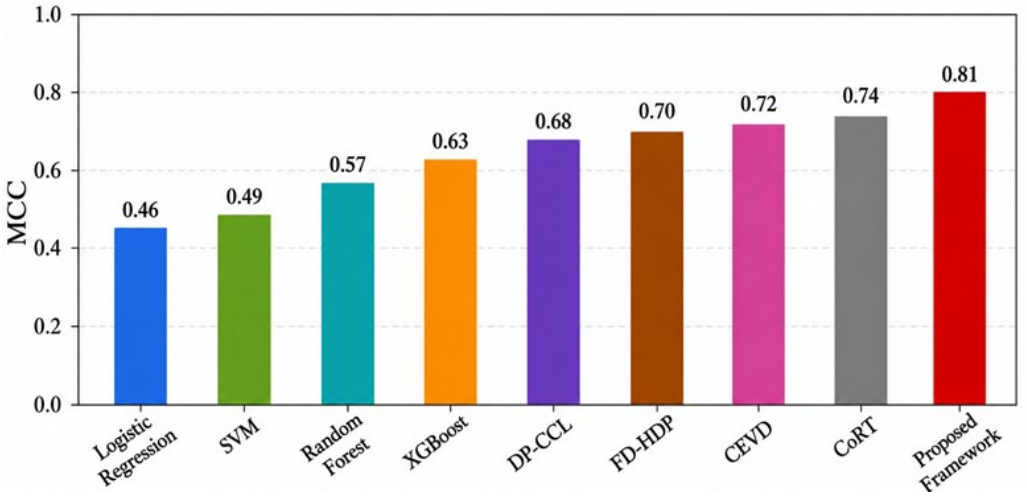

Figure 3: Comparison of Matthews Correlation Coefficient (MCC) values achieved by baseline approaches and the proposed framework.

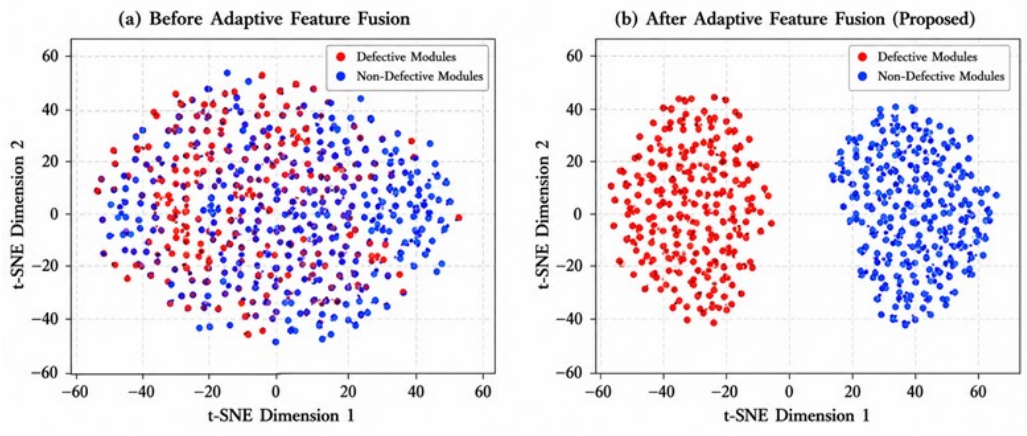

Figure 4: t-SNE projection of semantic embeddings learned before and after adaptive feature fusion.

4.7.3 Feature Space Distribution Analysis

To evaluate the effectiveness of the proposed semantic representation learning mechanism, t SNE visualization was performed on latent embedding representations generated by the framework. Figure 6 presents the feature distributions before and after adaptive feature fusion.

Before feature fusion, there was clear overlapping between the defective and non-defective clusters in the latent space, implying a low level of separability among features. Following the use of adaptive feature fusion and attention-based weight assignment, defective and non-defective clusters become clearly distinguishable from each other.

In Figure 4, the feature separability achieved shows that the Adaptive Feature Fusion Layer proposed in the model successfully improved the discriminating ability by combining semantic embedding and structural software metrics manually selected. With the application of attention on features, defect-sensitive information was highlighted, while irrelevant representations were minimized. The figure above illustrates that learning of project-invariant semantic representations was successfully carried out.

4.7.4 Cross-Project Domain Alignment Visualization

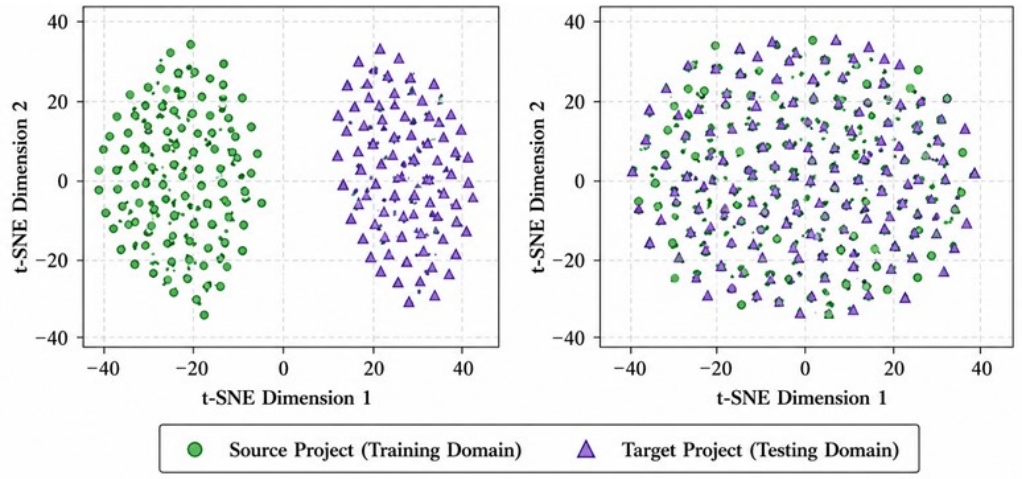

Figure 5: Visualization of latent feature distributions before and after cross-project domain alignment.

As shown in Figure 5, the influence of cross project feature alignment technique on the distribution of source and target domains can be seen. It is observed that there was substantial distributional difference between the source and target projects prior to alignment, owing to the differences between the domains. This usually leads to suboptimal results for Cross Project Software Defect Prediction techniques.

With the help of semantic representation alignment, the feature distributions of both the source and target domains became more aligned within the embedding space. This observation proves that the domain differences were successfully bridged through our proposed transfer learning approach.

Figure 5 clearly demonstrates the validity of the proposed transfer learning methodology for developing generalized defect representations.

4.7.5 Receiver Operating Characteristic (ROC) Analysis

Figure 6 displays the ROC curves for the proposed model and the traditional machine learning models. The proposed model performed better in terms of having the largest Area Under the Curve (AUC). This suggests that the proposed model had the best discriminative power between faulty and non-faulty software modules.

In comparison to the proposed model, the traditional machine learning models displayed relatively flat ROC curves due to difficulties in generalizing over different project settings.

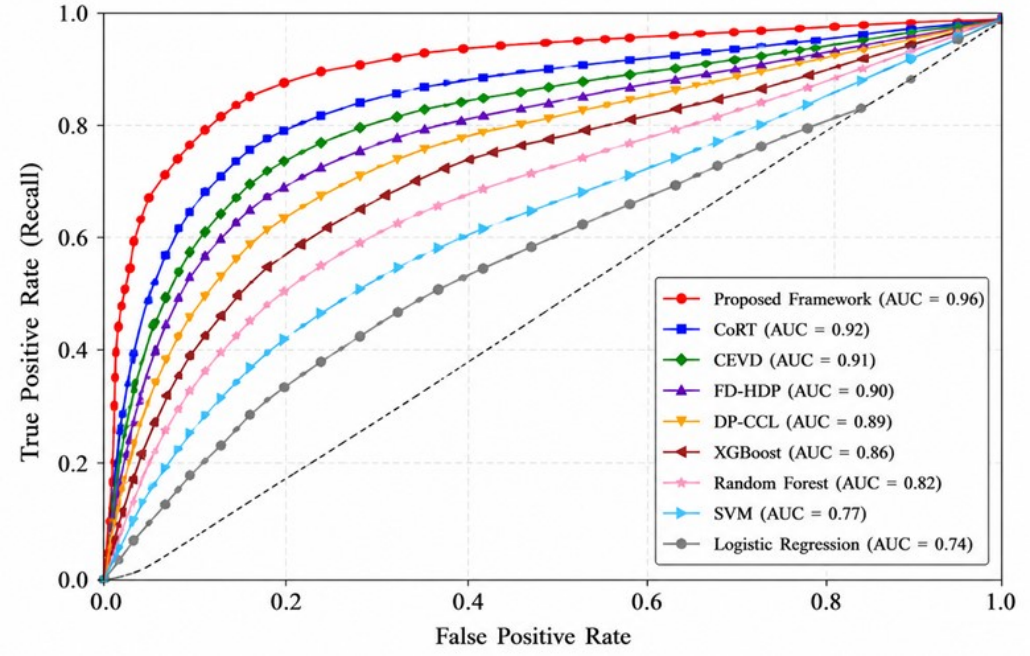

Figure 6: Receiver Operating Characteristic (ROC) comparison among evaluated prediction models.

This improved ROC performance shows that self supervised semantic learning and adaptive feature fusion enhanced classification sensitivity and prediction reliability in imbalance sensitive software defect prediction settings.

4.7.6 Ablation Analysis of Proposed Components

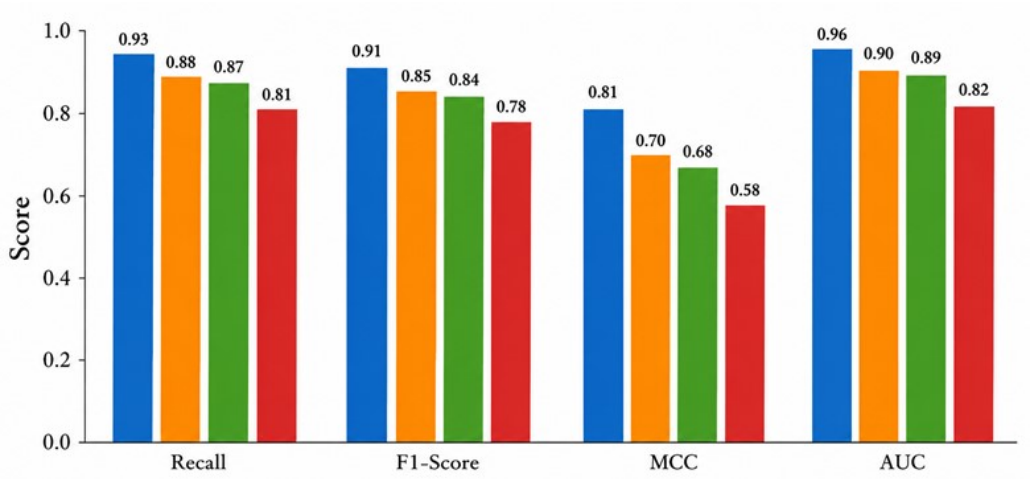

Figure 7: Ablation analysis showing the contribution of each major component within the proposed framework.

To evaluate the contribution of individual framework components, an ablation study was done using an incremental removal approach to major parts of the designed architecture. As shown in Figure 7 below, the impact on performance when contrastive learning, adaptive feature fusion, and cross-domain feature alignment were removed respectively can be observed. The blue bar indicates the full architecture, while the green, yellow, and red bars indicate architecture with no adaptive feature fusion, contrastive learning, and domain alignment respectively.

Accordingly, contrastive learning was seen to make the biggest drop in Recall and MCC, proving its importance in acquiring transferable semantic representation. Removal of adaptive feature fusion

made a major impact on feature discriminative capacity, whereas removal of domain alignment was seen to have had a negative impact on cross-project performance. From the analysis, the conclusion drawn is that all parts contribute positively towards improving performance.

4.7.7 Discussion of Visual Findings

From all the visualization results obtained in this paper, it is evident that the effectiveness of the proposed framework for Self-Supervised Semantic Representation Learning for CPD Prediction is verified. The convergence analysis proves that the optimization process of the framework is smooth, whereas the t-SNE plot shows that adaptive feature fusion enhances the separability of the semantic features. Moreover, from the cross-project alignment analysis, it is shown that the proposed framework can effectively decrease the domain difference between different software projects. The ROC curve analysis and ablation experiments prove the discriminative power and generalization performance of the proposed framework to be much better than those of previous baseline method.

5 Conclusion and Future Work

In this paper, we presented the Self-Supervised Semantic Representation Learning framework for CPDP. The self-supervised semantic representation framework leverages Transformer-based contrastive learning, adaptive feature fusion, and cross-project domain adaptation to learn transferable and defect-aware representations of software from diverse projects. In contrast with existing CPDP techniques that depend largely on manually engineered metrics as well as labeled software datasets, our proposed self-supervised semantic framework is capable of learning software representations through semantic learning without any manual intervention.

The experimental results on benchmark datasets including AEEEM and PROMISE datasets showed that the proposed framework outperforms the existing machine learning and the latest semantic learning baseline approaches in terms of different performance metrics. The self-supervised learning method yielded impressive performance with recall, F1 score, MCC, and AUC of 0.93, 0.91, 0.81, and 0.96 respectively, which demonstrates robustness and generalizability under the conditions of cross-project settings. Moreover, the Adaptive Feature Fusion Layer enhances the discrimination capability by combining handcrafted software metrics and semantic embeddings.

All in all, the results achieved indicate that it is possible to significantly enhance the quality of software defect prediction with the help of self-supervised semantic representation learning techniques under heterogeneous conditions, with a minimum number of data labeled manually required. As part of the subsequent research, the framework could be expanded by considering the following aspects: use of multimodal software representations (e.g., Abstract Syntax Trees, Code Property Graphs, and developer activity). Moreover, Graph Neural Networks and pretrained code models could also be considered. Similarly, future research may extend the proposed framework by incorporating lifecycle-aware temporal learning mechanisms similar to those utilized in ProfileSwitchNet [36, 37] to better capture evolving defect patterns across software repositories. In addition, adaptive optimization such as DCCP-based real-time architectures used by can be integration into self-supervised semantic representation learning for cross-project defect prediction as this integration is underexplored. Future research can also explore the assimilation of temporal-aware hybrid learning strategies inspired by electrical load forecasting models used in study [38] [39]to improve defect prediction in evolving software repositories. Upcoming studies can also investigate graph neural network-based semantic dependency modeling as used by [40] to further enhance cross-project defect prediction performance.

References

[1] N. Choubey, A. Cordero, J. P. Jaiswal, and J. R. Torregrosa, "Dynamical techniques for analyzing iterative schemes with memory," *Complexity*, vol. 2018, no. 1, p. 1232341, 2018.

[2] ——, "Dynamical techniques for analyzing iterative schemes with memory," *Complexity*, vol. 2018, no. 1, p. 1232341, 2018.

[3] A. Cordero, J. R. Torregrosa, and M. P. Vassileva, "A family of modified ostrowski's methods with optimal eighth order of convergence," *Applied mathematics letters*, vol. 24, no. 12, pp. 2082–2086, 2011.

[4] T. Lotfi, F. Soleymani, M. Ghorbanzadeh, and P. Assari, "On the construction of some tri-parametric iterative methods with memory," *Numerical Algorithms*, vol. 70, no. 4, pp. 835–845, 2015.

[5] J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.

[6] S. Kumar, V. Kanwar, S. K. Tomar, and S. Singh, "Geometrically constructed families of newton s method for unconstrained optimization and nonlinear equations," *International Journal of Mathematics and Mathematical Sciences*, vol. 2011, no. 1, p. 972537, 2011.

[7] J. F. Traub, *Iterative methods for the solution of equations*. American Mathematical Soc., 1982, vol. 312.

[8] A. Ostrowski, *Solution of Equations and Systems of Equations*, ser. Pure and applied mathematics. Academic Press, 1966. [Online]. Available: https://books.google.com.pk/books?id=uI2mAAAAIAAJ

[9] H.-T. Kung and J. F. Traub, "Optimal order of one-point and multipoint iteration," *Journal of the ACM (JACM)*, vol. 21, no. 4, pp. 643–651, 1974.

[10] R. F. King, "A family of fourth order methods for nonlinear equations," *SIAM journal on numerical analysis*, vol. 10, no. 5, pp. 876–879, 1973.

[11] J. Steffensen, "Remarks on iteration." *Scandinavian Actuarial Journal*, vol. 1933, no. 1, pp. 64–72, 1933.

[12] S. K. Mittal, S. Panday, L. Jäntschi, and L. C. Bolunduţ, "Two novel efficient memory-based multi-point iterative methods for solving nonlinear equations," *AIMS MATHEMATICS*, vol. 10, no. 3, pp. 5421–5443, 2025.

[13] S. Akram, M. Khalid, M.-u.-D. Junjua, S. Altaf, and S. Kumar, "Extension of king's iterative scheme by means of memory for nonlinear equations," *Symmetry*, vol. 15, no. 5, p. 1116, 2023.

[14] S. Akram, "On the construction of three step derivative free four $parametric methods with accelerated order of convergence,''$ 2015.

[15] A. Alazba, H. Aljamaan, and M. Alshayeb, "CORT: transformer-based code representations with self-supervision by predicting reserved words for code smell detection," *Empirical Software Engineering*, vol. 29, no. 3, p. 59, 2024.

[16] ——, "Automated detection of class diagram smells using self-supervised learning," *Automated Software Engineering*, vol. 31, no. 1, p. 29, 2024.

[17] H. H. Shehadeh, N. Aljarrah, R. A. Obeidat, and A. A. Abu-ein, "Attention-guided graph neural networks with adaptive feature selection for explainable software defect prediction," *Optimization and Information Computing*, vol. 15, pp. 2971–2995, 2026.

[18] Y. Bian, Y. Chen, H. Wang, S. Li, and Z. Cui, "Focus on what matters: Fisher-guided adaptive multimodal fusion for vulnerability detection," arXiv preprint arXiv:2601.02438, 2026.

[19] Z. Dong, Y. Guo, Q. Hu, Y. Zhang, M. Cordy, H. Liu *et al.*, "Learning generalizable multimodal representations for software vulnerability detection," arXiv preprint arXiv:2604.25711, 2026.

[20] W. Chen, J. Liu, K. Zhao, M. Zhang, and W. Chen, "CoverFL: Enhancing fault localization via deep coverage relation," *ACM Transactions on Software Engineering and Methodology*, 2026.

[21] C. Li, H. Li, Z. Li, M. Pan, and X. Li, "Enhancing fault localization in industrial software systems via contrastive learning," in *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, April 2025, pp. 691–703.

[22] Q. Zhang, Y. Shang, H. Hu, C. Fang, Z. Chen, and L. Xiao, "ComPass: Contrastive learning for automated patch correctness assessment in program repair," arXiv preprint arXiv:2602.07561, 2026.

[23] Y. Cao, Y. Dong, and J. Liu, "CEVD: Cluster-based ensemble learning for cross-project vulnerability detection," *Future Internet*, vol. 18, no. 2, p. 85, 2026.

[24] R. Haque, A. Ali, S. McClean, and N. Khan, "A zero-shot framework for cross-project vulnerability detection in source code," *Empirical Software Engineering*, vol. 31, no. 1, p. 3, 2026.

[25] E. Kotei and R. Thirunavukarasu, "A systematic review of transformer-based pre-trained language models through self-supervised learning," *Information*, vol. 14, no. 3, p. 187, 2023.

[26] Y. Zou and H. Wang, "A three-stage cross-project defect prediction framework based on feature representation and knowledge transfer," *Complex Intelligent Systems*, vol. 11, no. 11, pp. 1–21, 2025.

[27] M. Hesamolhokama, B. Rohani, A. Shafiee, M. Fazli, and J. Habibi, "From illusion to insight: change-aware file-level software defect prediction using agentic AI," arXiv preprint arXiv:2512.23875, 2025.

[28] H. Su, Z. Xu, Y. Zhang, and Q. Tan, "Source code vulnerability detection based on deep learning: a review," *Cybersecurity*, vol. 9, no. 1, p. 2, 2026.

[29] "Dataset for software defect predictions," https://figshare.com/articles/dataset/Dataset_For_Software_Defect_Predictions/19516858?file=34691257.

[30] "Augmented PROMISE dataset," https://zenodo.org/records/3309669.

[31] "Original PROMISE dataset," https://zenodo.org/records/268542.

[32] M. Shahid, M. A. Yahya, J. Song, H. M. Naveed, S. Yuksel, H. Dincer, and M. Ali, "Machine learning vs. traditional logistic regression: predictive performance and risk factor identification for child nutritional outcome in Pakistan," *BMC Public Health*, vol. 26, no. 1, p. 667, 2026.

[33] H. Huang and Q. Yang, "Large dimensional analysis of general margin based classification methods," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2021, no. 11, p. 113401, 2021.

[34] A. Hassan, V. Memon, F. B. Shaikh, S. Alahmari, M. Iqbal, and F. Azam, "Evaluating datagram congestion control protocol (DCCP) for real-time applications: A comparative study with TCP in multi-node networks," in *2024 International Conference on Engineering and Emerging Technologies (ICEET)*. IEEE, December 2024, pp. 1–6.

[35] P. Zhang, Y. Jia, and Y. Shang, "Research and application of XGBoost in imbalanced data," *International Journal of Distributed Sensor Networks*, vol. 18, no. 6, p. 15501329221106935, 2022.

[36] A. Hassan and M. Gulab, "ProfileSwitchNet: Predicting e-SIM carrier-switch behavior from provisioning and lifecycle signals," *Journal of Computational Informatics & Business*, vol. 3, no. 2, pp. 27–48, 2025.

[37] J. Wang, X. Sun, Q. Cheng, and Q. Cui, "An innovative random forest-based nonlinear ensemble paradigm of improved feature extraction and deep learning for carbon price forecasting," *Science of the Total Environment*, vol. 762, p. 143099, 2021.

[38] A. Hussain, S. T. Ahmed, and Z. Ahmed, "Electrical load prediction using statistical, deep learning, and hybrid time series models," *Journal of Computational Informatics & Business*, vol. 3, no. 2, pp. 49–62, 2025.

[39] I. Uzezi, P. Harcourt, and P. Harcourt, "Integrated machine-learning pipeline for software defect prediction in resource-constrained environments," vol. 020052, pp. 1–6, 2026.

[40] M. S. Salam and Z. Ahmad, "UrbanFlow-GST: A dynamic graph-based spatio-temporal fusion model for real-time traffic speed forecasting," *Journal of Computational Informatics & Business*, pp. 1–20, 2026.